

POLITECNICO DI TORINO

Master's Degree in Electronic Engineering



Politecnico di Torino

Master's Degree Thesis

Development of a monitoring system for non-rechargeable lithium batteries

Supervisors

Prof. Danilo DEMARCHI

Ph.D. Umberto GARLANDO

Ph.D. Alessandro SANGINARIO

Dott. Mattia BAREZZI

Candidate

Francesco CORIANÒ

April 2024

Summary

This thesis is an improvement of the electronic systems that compose WAPPFRUIT, a regional project focusing on smart technologies for water management in fruit growing, aiming to promote rural development in the Piedmont region. The project deployed systems to collect essential soil data (e.g., soil temperature, matric potential, volumetric water content) using LoRa network nodes to characterize and irrigate orchard fields. Efficient and long-lasting power supply is crucial for these systems. They rely on a primary battery in lithium-thionyl chloride (LiSOCl₂) chemistry without estimating the SOH (State of Health) battery. This chemistry is challenging in the monitoring of this parameter. When the battery is almost dead, there is a steep voltage drop phenomenon, making end-of-life detection vital in such a way as to prevent an electronic system for real-time monitoring from being switched off suddenly. This work has been addressed two main objectives. Firstly, firmware has been developed for managing battery end-of-life using the BQ35100 IC (Integrated Circuit) from Texas Instruments. Lastly, a PCB (Printed Circuit Board) has been designed to realize a new electronic prototype estimating LiSOCl₂ battery end-of-life. The research started with a deep analysis of BQ35100 manuals, revealing its complexity with RAM, flash memory, and internal algorithm for battery end-of-life assessment. Communication is established via the I2C (Inter-Integrated Circuit) protocol. Software development began with implementing the I2C command sequence to read the BQ35100's RAM, considering its timing constraints. The next phase involved the creation of a library for reading and writing data to flash memory, essential for LiSOCl₂ gauge operational mode. After the software development step, the challenge shifted to the selection of a suitable testing method to discharge the battery quickly and estimate the battery lifetime. Given the big battery capacity and the low maximum current discharge battery value, a full cycle could take months or years. A stress test solution has been designed to test BQ35100 functionalities and to observe the direct correlation between battery depletion and increasing internal battery resistance. In addition to this work, a PCB has been developed to monitor innovative electronic systems for precision agriculture, guaranteeing the maximum performance from the BQ35100 side. The thesis concluded by verifying effective battery end-of-life management.

This value will be extended beyond WAPFRUIT, offering an accurate way to estimate LiSOCl₂ battery lifetime in such a way as to realize long-lasting IoT (Internet of Things) devices for precision agriculture.

Acknowledgements

"Alla nonna Carmela e al nonno Uccio"

Non è semplice riassumere a parola tutta la mia gratitudine per le persone che sono qui con me a condividere questo traguardo importante.

Ai miei nonni che ormai non ci son più, ringrazio loro per il sostegno datomi durante tutto il mio percorso e questo traguardo è dedicato anche a loro.

Ai miei genitori Orazio e Lucia e a mio fratello Mattia che mi hanno assistito durante tutta la mia vita e soprattutto durante tutto questo lungo percorso universitario, perché nonostante tutto mi han sempre aiutato e supportato e senza di loro non sarei arrivato dove sono ora.

Ai miei amici “di giù e di su” che mi avete fatto passare momenti indimenticabili che probabilmente rimarranno con me per sempre.

A Ludovica per avermi supportato e incoraggiato in questo tempo insieme, nonostante le difficoltà incontrate non mi mai fatto perdere il sorriso o la voglia di andare avanti.

Table of Contents

List of Tables	VIII
List of Figures	IX
Acronyms	XII
1 Introduction	1
1.1 LoRaWAN Protocol	2
2 Technological Background	6
2.1 Analysis of the Battery: LS14500	6
2.2 Microcontroller with Transducer LoRa	8
3 System Design Steps	10
3.1 Battery Gauge	10
3.1.1 Measurements	11
3.1.2 Features	13
3.1.3 Operating Mode Choice	14
4 Firmware Development	16
4.1 Introduction	16
4.2 I ² C Characteristics	16
4.3 Type of commands	16
4.3.1 Data Commands	17
4.3.2 Control Subcommands	20
4.4 BQ35100 Functions Code	23
4.4.1 General Functions	23
4.4.2 Data Commands	27
4.4.3 Command MAC Subcommands	28
4.4.4 Memory Commands	30
4.5 Data Memory Mapping	32

5	Firmware Test	34
5.1	Previous Consideration for Test Procedure	34
5.1.1	Individual Command Test	35
5.2	Current Consumption Estimation	36
5.2.1	Python Code For Current Consumption	37
5.2.2	Estimation of Time to End	39
5.3	Possible Procedures	40
5.3.1	First Test Procedure	40
5.3.2	Second Test Procedure	41
5.4	Final Setup	42
5.5	Test Code	42
5.5.1	Starting project	42
5.5.2	Main function	43
5.5.3	LoRaWAN Protocol Functions	45
5.5.4	Additional Timer Elapse Routines	52
5.6	Decoder for TTN	56
6	Experimental Results	59
6.1	Data Managing	59
6.2	First Test	59
6.3	Second Test	62
7	Hardware Development	65
7.1	Components Required	65
7.2	Schematic	71
7.3	PCB Layout	73
7.4	Bills Of Materials	74
7.5	3D Models	75
7.6	Manufacturing	75
8	Conclusion and Future Perspective	77
	Bibliography	78

List of Tables

1.1	Maximum payload length in LoRa protocol considering The Things Network constrains.	3
7.1	Bill Of Materials.	74

List of Figures

1.1	WAPPFRUIT – Intelligent Technologies Applied to Water Management in Fruit Cultivation [1].	1
1.2	CSS Modulation of LoRa Protocol(Source [2]).	2
1.3	(LoRa class-A)Bidirectional end devices timing of transmission/reception cycle(Source [2]).	4
1.4	(LoRa Class-B) Bidirectional end devices with scheduled receive slots timing of transmission/reception cycle(Source [2]).	4
1.5	(LoRa Class-C) Bidirectional end devices with maximal receive slots timing of end device timing of transmission/reception cycle(Source [2]).	5
1.6	Protocol stack of LoRaWAN(Source [2]).	5
2.1	LS14500 AA Size Battery	6
2.2	Typical LS14500 discharge profiles at + 20°C(Source [3]).	7
2.3	Voltage plateau versus Current and Temperature (at mid-discharge) [3].	8
2.4	Restored Capacity versus Current and Temperature with 2.0 V cut-off [3].	8
2.5	STM32-WL55JC1 Nucleo Board from STMicroelectronics (Source [4]).	9
3.1	BQ35100 Single-Cell Simplified Implementation [5].	11
3.2	Functional Block Diagram of BQ35100 [5].	12
3.3	Current consumption of BQ35100 according to datasheet [5].	14
3.4	EOS Usage Diagram from [7].	15
3.5	Scaled Resistance Profile of LS14500 Battery with EOS Flag Thresholds from [8].	15
4.1	I ² C-Compatible Interface Timing Characteristics from datasheet [5].	17
4.2	Data Command Summary.	18
4.3	Example I ² C Data Command Transition Sequence.	20
4.4	Control MAC Subcommands Summary.	21

4.5	Example I ² C Control Subcommand Transition Sequence.	23
5.1	Control_Status Command Example	35
5.2	Consumption Monitored From System Monitoring Soil Water Content [11].	36
5.3	Consumption Monitored From Drip Irrigation System [12].	37
5.4	Setup Configuration for Testing Procedure.	42
5.5	Flow Chart of Testing Procedures.	43
6.1	TTN decoded samples from TTN console network.	59
6.2	Voltage vs Time (First Test).	60
6.3	State Of Health Behaviour vs Time (First Test).	61
6.4	Internal Resistance Behaviour vs Time (First Test).	61
6.5	State Of Health Behavior During Time (Second Test).	62
6.6	Voltage Behaviour respect to Time and Depth-of-Discharge (Second Test).	63
6.7	Internal Resistance Behaviour respect to Time and Depth-of-Discharge (Second Test).	64
7.1	3D Model of LoRaTO Module.	65
7.2	BQ35100-PWR Chip.	66
7.3	HDC3022 Temperature and Humidity Sensor Chip (Source [16]).	66
7.4	STQL020C33R Voltage Regulator (Source [17]).	68
7.5	SIP32431 P-channel Pass Transistor (Source [18]).	68
7.6	Example of SMD Capacitors.	69
7.7	1024TR AA-size Battery Holder.	69
7.8	1210L Series PTC RESET (Source [19]).	70
7.9	LM66100, Low IQ Ideal Diode With Input Polarity Protection.	70
7.10	Schematic Sheet.	72
7.11	Top and Bottom View with Highlighting on Different Zones.	73
7.12	Top 3D view and Bottom 3D view of the PCB.	75
7.13	Top view and Bottom view of the Frost Detection System assembled.	76

Acronyms

ACK

Acknowledge

ADC

Analog to Digital Converter

DoD

Depth of Discharge

EOS

End Of Service

GE

Gauge Enable

IoT

Internet of Things

I²C

Inter-Integrated Circuit

LoRa

Long Range protocol - physical layer

LoRaWAN

Long Range Wide Area Network - networking layers

MCU

Micro Controller Unit

MAC

Manufacturing Access Control

NACK

Not Acknowledge

SCL

Serial Clock

SDA

Serial Data

SOH

State Of Health

TRM

Technical Reference Manual

TTN

The Things Network

Chapter 1

Introduction

This thesis project is the prosecution of project WAPPFRUIT - Smart technologies applied to water management in fruit growing for rural development in Piedmont. The main goal of WAPPFRUIT is to innovate the fruit farming through the use of state-of-the-art technologies that enable the determination of the correct water requirements and the complete automation of the micro-irrigation system utilizing available market sensors to measure the soil matric potential, thereby indirectly determining the water needs of the plants.

The data are collected at regular time intervals (preset by the user) using a control unit and will be remotely available using through a web interface and a dedicated smartphone app allowing the user to monitor in real-time the field conditions.

To achieve the goal have an efficient and long-lasting power supply for the control unit is needed and for this aim the most suitable choice was to employ an Lithium-Thionyl Chloride (LiSOCl_2) primary battery.

So it becomes clear that the management of the battery conditions is a crucial aspect in WAPPFRUIT that must be taken into account. This is the main goal of this work: to develop a system to manage the Lithium-Thionyl Chloride battery



Figure 1.1: WAPPFRUIT – Intelligent Technologies Applied to Water Management in Fruit Cultivation [1].

that allows the user to have in real-time messages with the status of the battery and to be alerted at a settled point before the end-of-life battery condition.

It's clear that all developed systems among WAPFRUIT are IoT compliant, which means that each system is a part of a network of physical objects (or "things") that are embedded system that can communicate and exchange data with other devices and system over the internet.

So each system of WAPFRUIT uses a LoRa gateway, the stakeholder that gathers data from all nodes, that is the connection between the local LoRa network and the internet.

In the following section a little introduction of LoRaWAN protocol is shown along with some theoretical concepts associated with it. This will help in understanding certain discussions that will be covered in the subsequent chapters.

1.1 LoRaWAN Protocol

LoRa technology is one of the Low Power Wide Area Network (LPWAN) technologies available on the market. This technology uses radio frequency bands in the sub-gigahertz range without licensing, each carrier waves different in each terrestrial zone, for example in Europe the carrier waves used are at 169 MHz, 433 MHz, 868 MHz.

LoRa technology uses Chirp Spread Spectrum (CSS) modulation technique, in which to code the information each pulse is modulated in frequency linearly in a wideband. If the frequency increases in time it is called up-chirp and if the frequency decreases linearly in time it is called downchirp. The frequencies are shown in Figure 1.2.

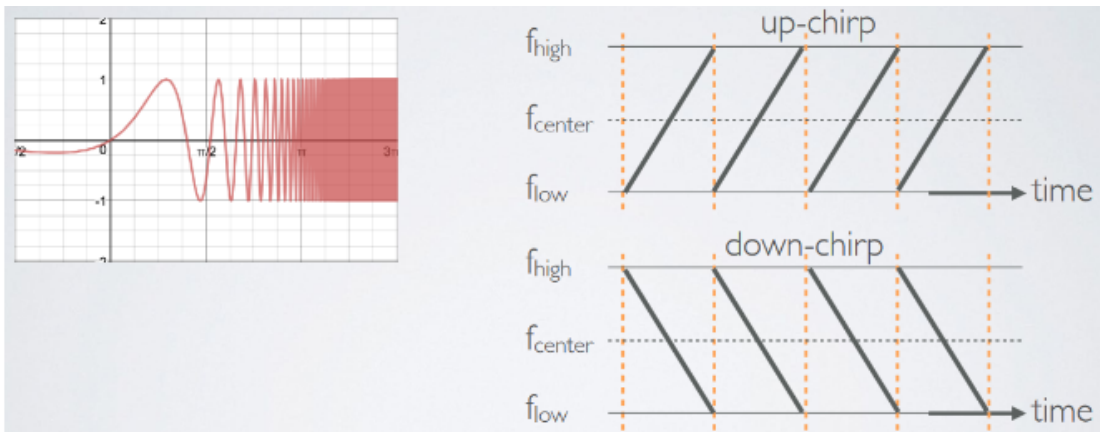


Figure 1.2: CSS Modulation of LoRa Protocol(Source [2]).

This technique enables long-range transmission up to 2 to 5 km in urban regions and 10 km in rural regions using a very low amount of energy.

This technology can be employed for end-to-end communication in which two devices, called nodes, can exchange data without any supporting appliances or exchange data from some devices to the internet thanks to a gateway which collects data from each LoRa device in its area of action and sends them to a LoRa provider via LAN or WLAN.

In order to classify a radio protocol the bit rate (R_b) must be evaluated in relation to the CCS modulation variables:

$$R_b = SF \cdot \frac{1}{2^{SF}} \cdot BW \text{ bits/sec,}$$

where BW is the modulation bandwidth (125 kHz, 250 kHz or 500 kHz), the interval of frequencies where the signal is up-chirped or down-chirped centered on the carrier frequency, and SF is the spreading factor that can change from 7 to 12 and it defines how many bits are encoded in a single chirp.

So, fixing a certain slot of time, higher values of SF means also higher airtime on the transmission medium.

In Table 1.1 are reported all typical values of data rate for EU868.

Data Rate	Configuration	Bandwidth	Bits/s	Maximum Data Payload
DR0	SF12	125kHz	250	51
DR1	SF11	125kHz	440	51
DR2	SF10	125kHz	980	51
DR3	SF9	125kHz	1760	115
DR4	SF8	125kHz	3125	222
DR5	SF7	125kHz	5470	222
DR6	SF6	125kHz	11000	222
DR7	FSK:50kpbs	50kbps	50000	222

Table 1.1: Maximum payload length in LoRa protocol considering The Things Network constrains.

As shown, LoRa uses six possible spreading factors (SF7 to SF12) to adapt the data rate and range tradeoff. Higher spreading factors allows longer range at the expense of lower data rate, and vice versa.

In addition to these rules, each LoRa provider provides some kind of fair access policy in such a way that each end device can send data with a 30 seconds uplink airtime, per 24 hours, per device and at most 10 downlink messages per 24 hours, including the ACKs for confirmed uplinks as reported on The Things Network website.

LoRaWAN provides various classes of end devices to address different requirements in IoT appliances.

- Class-A: allows bidirectional communication whereby each uplink transmission (from an end device) is followed by two short downlink reception windows, as shown in Figure 1.3.

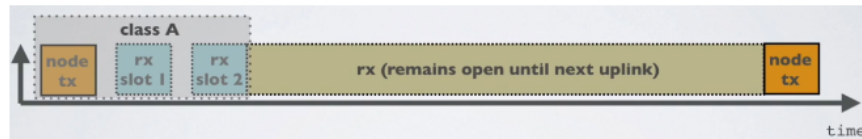


Figure 1.3: (LoRa class-A) Bidirectional end devices timing of transmission/reception cycle (Source [2]).

These transmission slots are scheduled by the end device based on its own communication needs (distance, power consumption...). This class is the lowest power end-device system for applications that only requires short downlink communication after the end device has sent an uplink message.

- Class-B: they have the same timing as class-A where these devices open extra receive windows at the scheduled time. To open these receive windows at the scheduled time, end devices receive a time-synchronized beacon from the base station. In this way, the network server knows when the end device is listening.

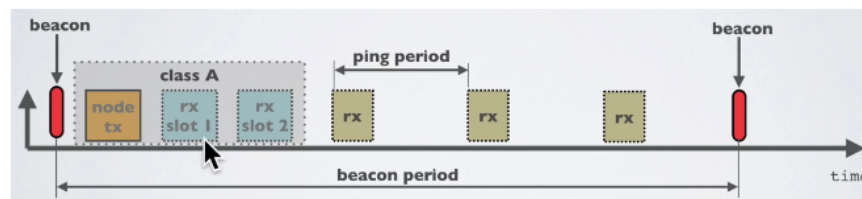


Figure 1.4: (LoRa Class-B) Bidirectional end devices with scheduled receive slots timing of transmission/reception cycle (Source [2]).

- Class-C: have almost continuously open RX windows and only close when transmitting. This guarantees low latency at the expense of greater energy consumption.



Figure 1.5: (LoRa Class-C) Bidirectional end devices with maximal receive slots timing of end device timing of transmission/reception cycle(Source [2]).

It is important to clarify that these classes are defined by the protocol where the first class (class-A) is the most common, class-B is on the market but less common, and class-C is still under development. The result is a so-called LoRa protocol stack that is shown in Figure 1.6.

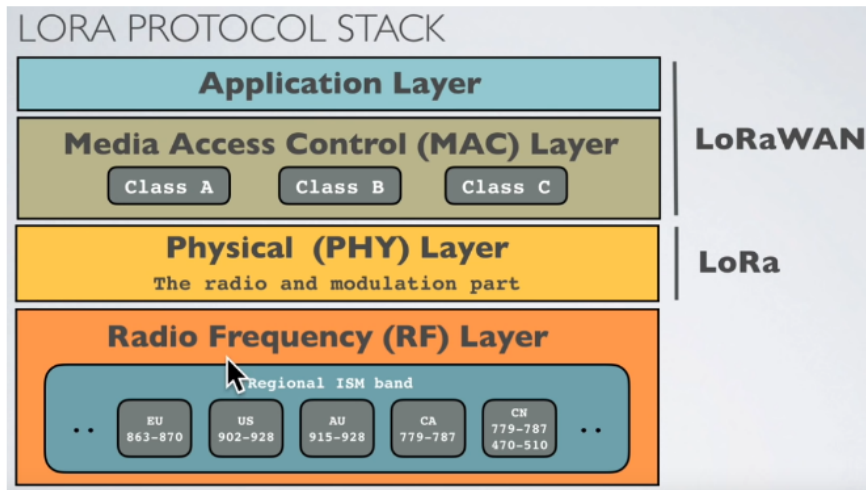


Figure 1.6: Protocol stack of LoRaWAN(Source [2]).

Chapter 2

Technological Background

2.1 Analysis of the Battery: LS14500

The subject of examination is the LS14500 AA battery manufactured by Saft (Figure 2.1), identified as a lithium thionyl chloride (LiSOCl₂) battery. According to the information provided in the datasheet [3], this battery has a self-discharge rate of 1% per year of storage at 20 °C. Additionally, it is characterized by high capacity and energy density, resulting in a nominal capacity of 2.6 Ah. Operating at a standard voltage of 3.6 V and being AA-sized, it also demonstrates resistance to corrosion and exhibits an extensive operating temperature range from -60 °C to 85 °C.

Of significant note is its nominal voltage, which offers practical advantages by necessitating only a low-dropout regulator to uniformly distribute the supply voltage across the entire system at 3.3 V, a widely used nominal voltage for low-voltage electronic devices. Furthermore, throughout its entire life cycle, the discharge profile of this battery maintains a consistently flat voltage, as depicted in Figure 2.2. When the resistance load is high (so, the output current is in the order of a few mA), as shown by red, blue, green lines, it is always in the proximity of a cell voltage of 3.4 V-3.6 V. This is a good benefit when the system is operative but declares some issues in the estimation of when the battery is at end-of-life condition and should be replaced.

As partial drawbacks of this type of battery, it is useful to highlight the fact that it has a very high output impedance. The chemical reaction that enables extremely low self-discharge and long shelf life (passivation formation) have the unwanted effect of limiting the available output current:

- a maximum continuous output current of 50 mA;



Figure 2.1:
LS14500 AA
Size Battery

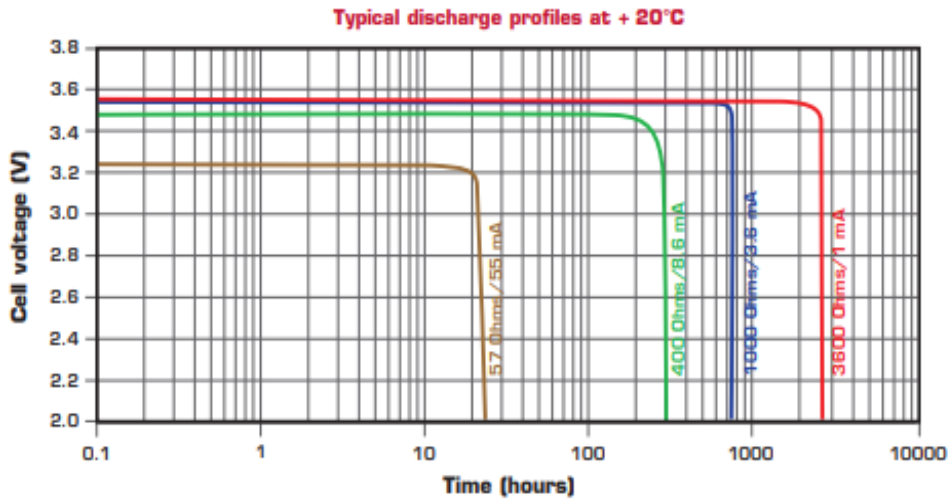


Figure 2.2: Typical LS14500 discharge profiles at + 20°C(Source [3]).

- a maximum peak output current up to 250 mA for 0.1 sec.

Those values are definitively low with respect to other non-rechargeable batteries. But in real case scenario of an smart-agriculture system this is not a big problem because the current consumption in LoRa transmission/reception is lower respect to the maximum continuous current output limit (45mA) and lasts few seconds, but this can be an issue when it comes to test an entire life cycle of the battery. This issue will be better discussed later in chapter 5.

Also another drawbacks to take into account is the fact that the LS14500, as all other lithium-ion based batteries, has differences in the voltage and the capacity when a load is connected respect to the nominal values.

In particular the behaviour of both battery voltage is show in figure 2.3 and battery capacity is shown in figure 2.4. As shown:

- the value of the battery voltage decreases respect to the nominal value when a higher consumption is requested from the battery, so to have a value close to the nominal battery voltage value a consumption near 1mA is needed;
- the battery capacity has an initial lower value, increasing when the consumption is between 1mA and 2mA and decreasing constantly after 2mA.

Having said this, these considerations must be taken carefully into account when the battery is used inside a of smart agriculture systems, so a proper sub-system inside of it which deals with the management of battery life and the prediction of the end-of-life condition is needed.

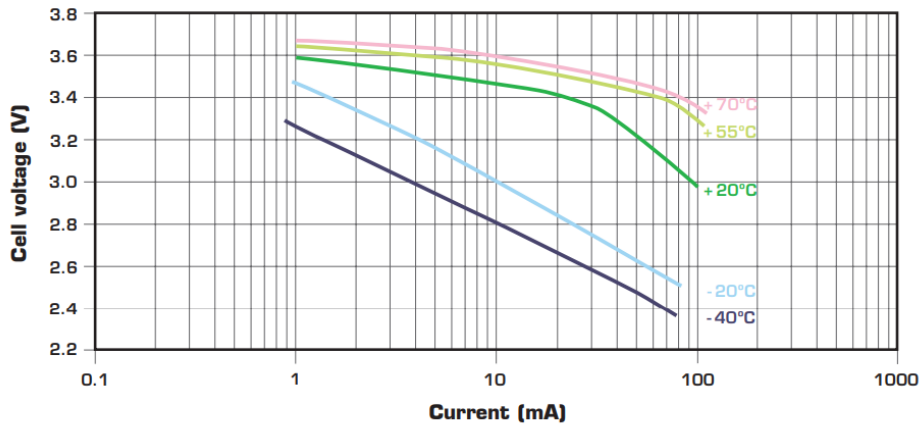


Figure 2.3: Voltage plateau versus Current and Temperature (at mid-discharge) [3].

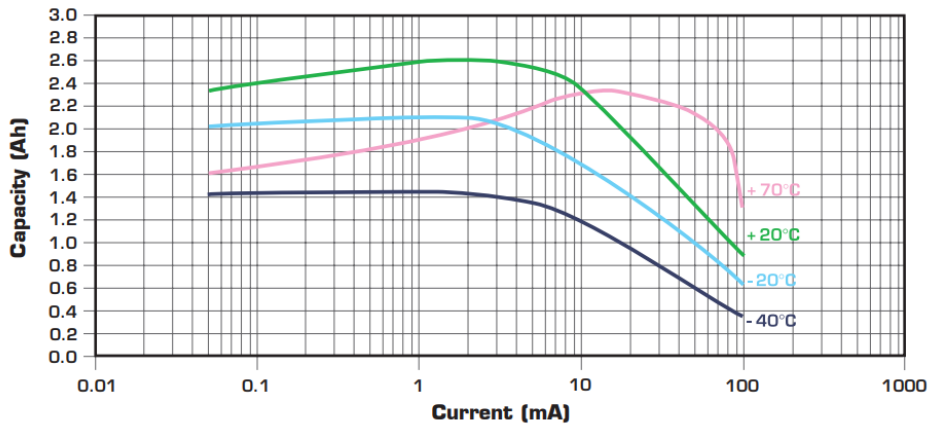


Figure 2.4: Restored Capacity versus Current and Temperature with 2.0 V cut-off [3].

2.2 Microcontroller with Transducer LoRa

The MCU used in the development of the system is the STM32-WL55JC1 [4] that is required to handle the communication between the system and the gateway using the LoRaWAN protocol, to synchronize each operation properly. It is a dual core ultra-low-power MCU with an embedded radio module handling the LoRa protocol. The figure of the Nucleo board is shown in figure 2.5.



Figure 2.5: STM32-WL55JC1 Nucleo Board from STMicroelectronics (Source [4]).

Chapter 3

System Design Steps

The design steps are based of the characteristic of the battery under consideration and the STM32-WL55JC1 MCU.

The needed solution must be able to detect the battery end-of-life condition while also have some way to communicate with the MCU, in order to extract data that an host would willing to have available for any kind of use.

3.1 Battery Gauge

As discussed in previous section the characteristic of the LS14500 primary battery make difficult to predict the end-of-life condition of the battery itself, because of the very flat behaviour of the voltage.

The optimal solution to achieve the detection of end-of-life condition is to use the BQ35100 integrated circuit from Texas Instrument, that is specially designed for lithium-ion based non-rechargeable batteries. A basic usage of the gauge is shown in figure 3.1.

The BQ35100 gauge is an integrated circuit suitable for monitoring the conditions of the battery regardless the chemical type of lithium battery.

The working principle can be explained seeing the Figure 3.2 recorded from its datasheet [5], that shows the functional block diagram of the BQ35100.

As reported in the datasheet [5], the BQ35100 is a complex integrated circuit that has inside:

- Internal Oscillator with an operating frequency of 2MHz;
- Integrating Analog to Digital Converter (ADC) providing voltage and temperature measurements;
- Coulomb Counter providing current measurements using a sense resistor between SRN and SRP pin;

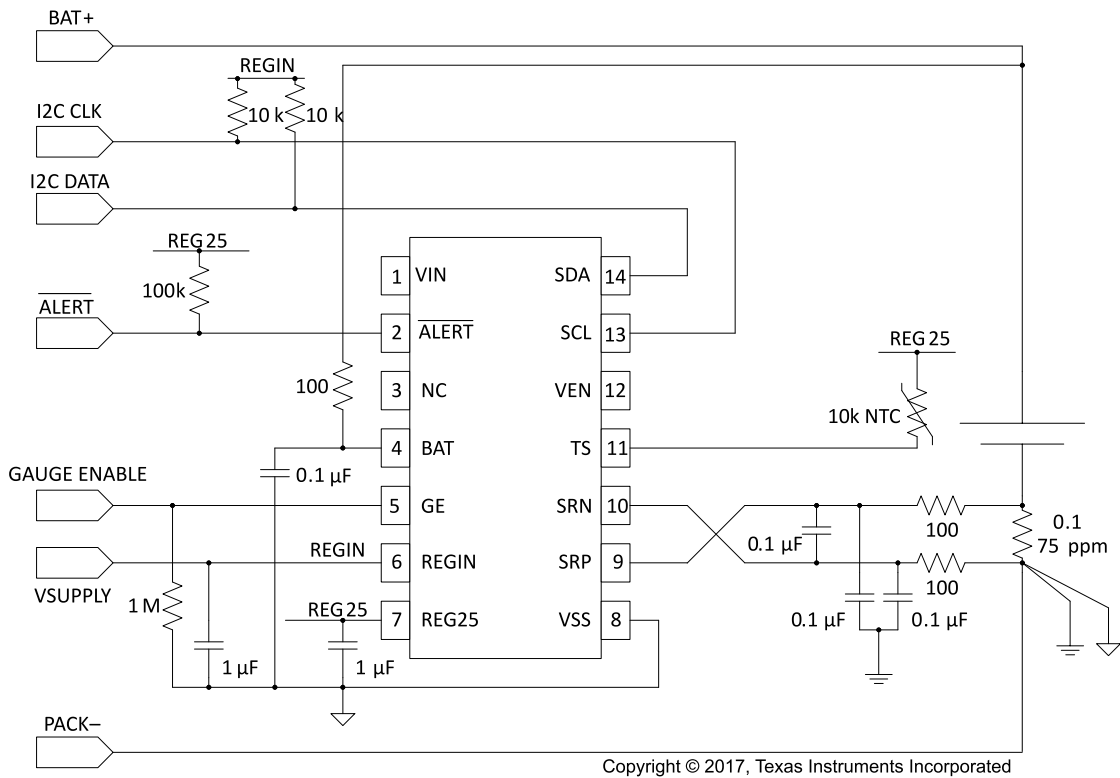


Figure 3.1: BQ35100 Single-Cell Simplified Implementation [5].

- Internal Temperature Sensor;
- Data Memory used to program the gauge according to the features needed based on the application;
- Gauge Enable digital pin used to enable the gauge and the internal LDO;
- Linear Voltage Regulator with 2.5V output voltage;
- Peripheral Unit with a \overline{ALERT} signal that can be configured to be triggered by a variety of status conditions;
- Internal Processing Unit responsible for applying the Gauging Algorithm in order to determine SOH and EOS of the battery;
- I²C Interface that allows the communication with an external MCU unit.

3.1.1 Measurements

The BQ35100 is capable to make different measures that it internally uses in the gauge algorithm to determine the when the battery is near at the end-of-life.

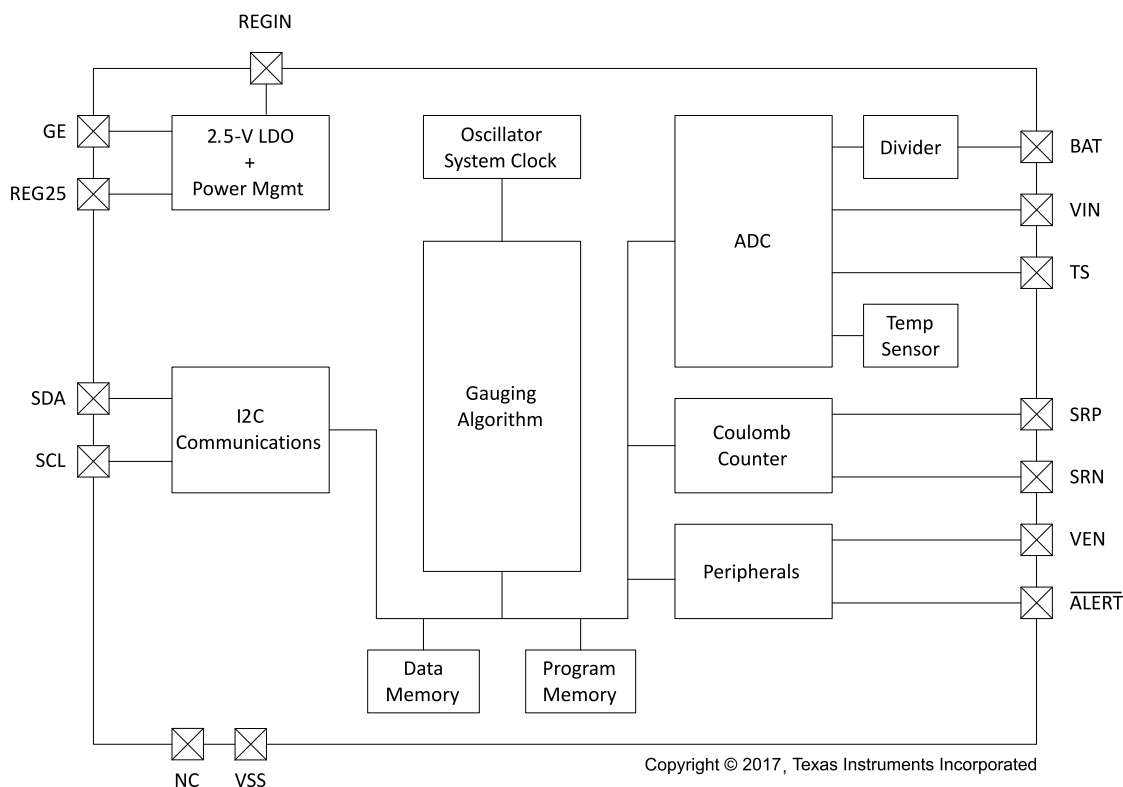


Figure 3.2: Functional Block Diagram of BQ35100 [5].

Among the measurements the battery gauge can measure:

- Current: it measures the discharge current of the battery by measuring the voltage drop across a small-value sense resistor between the SRP and SRN pins using the integrating delta-sigma ADC in the device;
- Battery input Voltage: using the same delta-sigma ADC, which at will can be scaled by the internal translation network, whose translation gain function is determined during calibration process;
- Temperature: it can use either
 - a) Internal Sensor with a voltage gain $G_{TEMP} = -2mV/^\circ C$;
 - b) External NTC thermistor.

The choice between the two is made by setting the corresponding pin in the right cell in data memory during configuration process.

According to the datasheet [5] the gauge must be first calibrated following the calibration procedures listed in the TRM [6]. Each calibration routine follows the next procedure:

1. Enter calibration mode;
2. Enter the known value of the quantity that must be calibrated;
3. Measure the designed quantity (temperature, voltage or current);
4. Evaluate the difference between the inserted value and the measured one;
5. Save the parameters calculated in the dedicated memory cell;
6. Exit calibration mode.

If the calibration is not performing well, than that will be a more considerable error in data collected.

3.1.2 Features

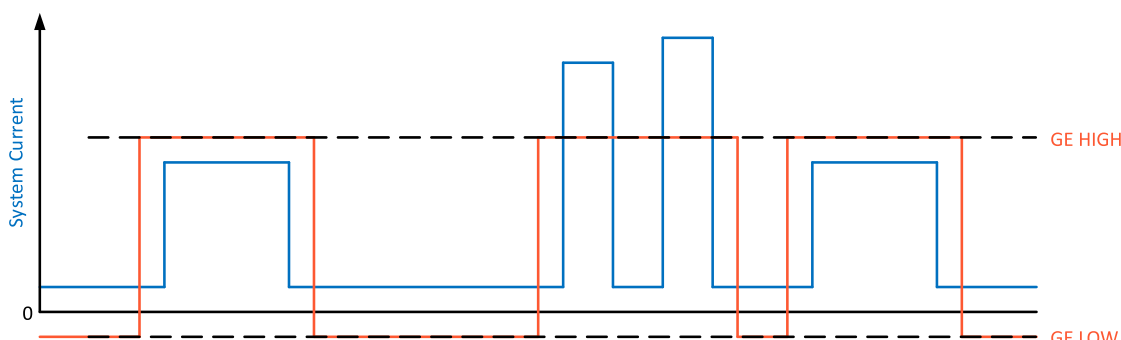
As said before the BQ35100 battery gauge has three different operating modes that can be use to determine the status of any kind of battery. Each one of the mode can be selected by triggering the righth bits in a specific memory cell. The behaviour of each mode is reported below like in the datasheet[5] and TRM[6]:

1. Accumulator Mode (ACC): this mode is suitable for any kind of battery when the capacity of the battery is known. In this mode, the BQ35100 device measures and updates cell voltage, cell temperature, and load current every 1s. To begin accumulation, the gauge must be put in active mode, and when accumulation ends, the gauge must be put in stop mode. Then the gauging algorithm evaluate the accumulated charge used and compared it with the full capacity of the battery, which determine the Depth-Of-Discharge.
2. State Of Health Mode (SOH): this mode is suitable for determining SOH for lithium manganese dioxide (Li-MnO₂) chemistry. In this mode, cell voltage and temperature are measured immediately after the GE pin is asserted. The gauge uses this data to compute SOH. Once the initial update occurs and the host reads the updated SOH, then the device can be powered down.
3. End Of Service Mode (EOS): this mode is suitable for gauging lithium thionyl chloride (Li-SOCl₂) cells. The end-of-service (EOS) gauging algorithm uses voltage, current, and temperature data to determine the resistance and rate of change of resistance of the battery. The resistance data is then used to find Depth of Discharge (DOD) = DOD(R). As above, SOH is determined and in turn used to determine the EOS condition, but it can happen that SOH is wrong depending on various factor, so in this mode it is not reliable.

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
$I_{CC_ACCU}^{(1)}$	Gas gauge in ACCUMULATOR mode	GE = High AND <i>GaugeStart()</i> received and <i>GaugeStop()</i> not Received (GMSEL1,0 = 0,0)		130		μA
$I_{CC_SOH}^{(1)}$	State-of-health operating current	GE = High AND <i>GaugeStart()</i> received and <i>GaugeStop()</i> not Received (GMSEL1,0 = 0,1)		40		μA
$I_{CC_EOS_Burst}^{(1)}$	End-of-service operating current—data burst	GE = High AND <i>GaugeStart()</i> received and <i>GaugeStop()</i> not Received (GMSEL1,0 = 1,0)		315		μA
$I_{CC_EOS_Gather}^{(1)}$	End-of-service operating current—data gathering	GE = High AND <i>GaugeStart()</i> AND <i>GaugeStop()</i> Received (GMSEL1,0 = 1,0)		75		μA
$I_{CC_GELOW}^{(1)}$	Device Disabled	GE = LOW		0.05		μA

(1) Not production tested

(a) BQ Modes Current Consumption.



(b) Current profile of BQ35100.

Figure 3.3: Current consumption of BQ35100 according to datasheet [5].

The BQ35100 datasheet also provides the consumption of the gauge accordingly to the operational mode used. Those values are shown in figure 3.3. The datasheet [5] shows another important characteristic of this battery gauge: a very low battery consumption both in active mode ($300\mu A$ maximum in EOS mode data-burst) and in disable mode ($0.05\mu A$).

3.1.3 Operating Mode Choice

Accordingly to the battery chemical and discharge characteristic of the battery LS14500 the most suitable mode to use the BQ35100 is the EOS mode, that allows to related the internal resistance of the battery to the end-of-life condition of the battery.

The gauge algorithm uses temperature, current and voltage measurements in order to allow the gauge to evaluate the internal resistance of the battery and to detect when the battery reach a certain DOD.

The cycle of the EOS learning procedure is shown in figure 3.4.

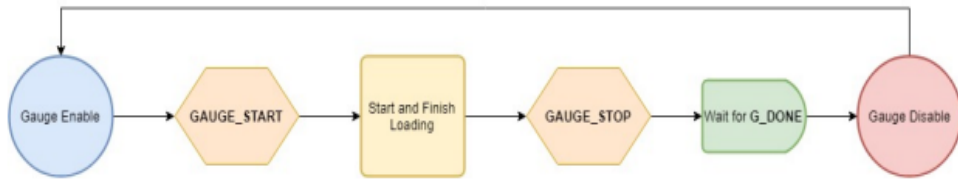


Figure 3.4: EOS Usage Diagram from [7].

An example of the behaviour of the internal resistance of the battery with respect to time is presented in figure 3.5

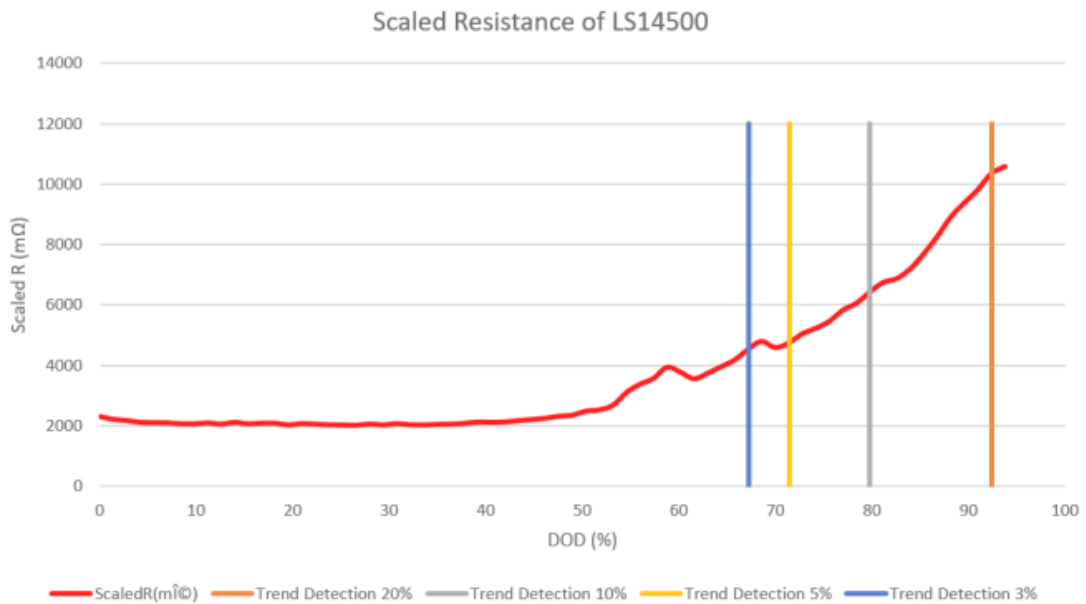


Figure 3.5: Scaled Resistance Profile of LS14500 Battery with EOS Flag Thresholds from [8].

Chapter 4

Firmware Development

4.1 Introduction

The main part of the project is to develop a library with commands that can exploit every functionality of the BQ35100 that use the I²C protocol in order enable the communication between the the MCU and the gauge following the requirements reported in the datasheet [5].

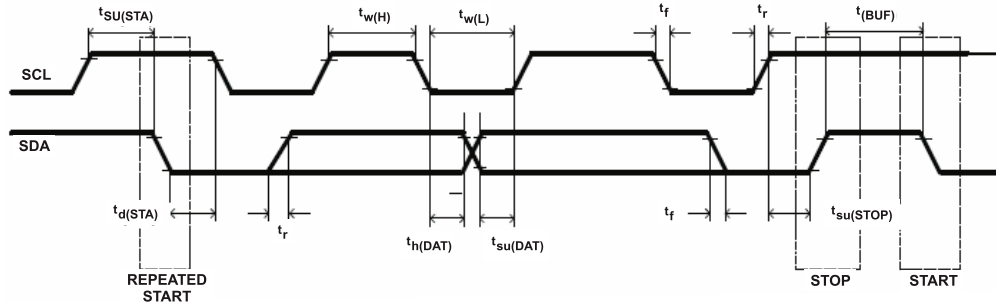
4.2 I²C Characteristics

In order to do so the first step is to understanding properly the characteristics of the I²C of the gauge in terms of conditions on timing.

The timing characteristics are shown in Figures 4.1a and Figure 4.1b taken form the datasheet of the gauge [5]: The timing constrains don't just put conditions on how the firmware have to be written, but also they put a restriction on the values of the pull-up resistors used on SCL and SDA lines. Those considerations are covered in chapter 7 related to the PCB development and they will not discussed in this chapter anymore.

4.3 Type of commands

The BQ35100 chip has two different types of command that are related that differs one another from the length of the command, the type of the command (red-only/write-only) and the procedure that must be followed in order for the command to be properly interpreted by the gauge.



(a) I²C-Compatible Interface Timing Diagrams.

		MIN	NOM	MAX	UNIT
t_R	SCL/SDA rise time			300	ns
t_F	SCL/SDA fall time			300	ns
$t_{W(H)}$	SCL pulse width (high)	600			ns
$t_{W(L)}$	SCL pulse width (low)	1.3			μ s
$t_{SU(STA)}$	Setup for repeated start	600			ns
$t_{d(STA)}$	Start to first falling edge of SCL	600			ns
$t_{SU(DAT)}$	Data setup time	100			ns
$t_{h(DAT)}$	Data hold time	0			ns
$t_{SU(STOP)}$	Setup time for stop	600			ns
t_{BUF}	Bus free time between stop and start	66			μ s
f_{SCL}	Clock frequency			400	kHz

(b) I²C-Table Parameters: $T_A = -40^\circ C$ to $85^\circ C$, $2.45 V < V_{REGIN} = V_{BAT} < 5.5 V$; Typical Values at $T_A = 25^\circ C$ and $V_{BAT} = 3.6 V$ (unless otherwise noted).

Figure 4.1: I²C-Compatible Interface Timing Characteristics from datasheet [5].

4.3.1 Data Commands

These commands are the most simple ones and so the perfect choose to begin with. The Data Commands are related to the data collected from the sensors and available trough the RAM of the BQ35100. All Data Commands are reported in figure 4.2, shown in both datasheet [5] and TRM [6].

As shown in the table, this kind of command is read-only command with some exceptions:

- 0x00 Control: Issuing a Control() (or Manufacturer Access Control or MAC) command requires a 2-byte subcommand;
- 0x02 AccumulatedCapacity: This read-word 4-byte command returns the accumulated coulombs since the coulomb counter was started. It provides an unsigned integer value with the range of 0 to 4.29E9 μ Ah. If the value reaches full it will hold at the full count and not roll over;
- 0x06 Temperature: This read-only command pair returns an unsigned integer

Cmd	Mode	Name	Format	Size in Bytes	Min Value	Max Value	Default Value	Unit
0x00...0x01	R/W	<i>Control</i>	Hex	2	0x00	0xff	—	—
0x02...0x05	R	<i>AccumulatedCapacity</i>	Integer	4	0	4.29e9	—	μAh
0x06...0x07	R	<i>Temperature</i>	Signed Int	2	-32768	32767	—	0.1 K
0x08...0x09	R	<i>Voltage</i>	Integer	2	0	65535	—	mV
0x0A	R	<i>BatteryStatus</i>	Hex	1	0x00	0xff	—	—
0x0B	R	<i>BatteryAlert</i>	Hex	1	0x00	0xff	—	—
0x0C...0x0D	R	<i>Current</i>	Signed Integer	2	-32768	32767	—	mA
0x16...0x17	R	<i>Scaled R</i>	Integer	2	0	65535	—	mΩ
0x22...0x23	R	<i>Measured Z</i>	Integer	2	0	65535	—	mΩ
0x28...0x29	R	<i>InternalTemperature</i>	Signed Integer	2	-32768	32767	—	0.1 K
0x2E...0x2F	R	<i>StateOfHealth</i>	Integer	1	0	100	—	%
0x3C...0x3D	R	<i>DesignCapacity</i>	Integer	2	0	65535	—	mAh
0x79	R	<i>Cal_Count</i>	Hex	1	0x00	0xff	—	
0x7a...0x7B	R	<i>Cal_Current</i>	Signed Int	2	0	65535	—	mA
0x7C...0x7D	R	<i>Cal_Voltage</i>	Integer	2	0	65535	—	mV or Counts ⁽¹⁾
0x7E...0x7F	R	<i>Cal_Temperature</i>	Integer	2	0	65535	—	K

Figure 4.2: Data Command Summary.

value of the temperature, in units of 0.1°K, measured by the device and has a range of 0 to 6553.5°K. The source of the measured temperature is configured by the [TEMPS] bit in Operation Config A;

- 0x08 Voltage: This read-word command pair returns an unsigned integer value of the measured battery voltage in mV with a range of 0 V to 65535 mV;
- 0x0A BatteryStatus: This read-only register provides indications on the status of the battery;
- 0x0B BatteryAlert: This read-only register provides indications on the cause of the ALERT pin trigger. An ALERT bit only clears if the condition for it is removed. Reading this register causes the ALERT pin to deassert and also clears the ALERT bit in BatteryStatus(). Note the ALERT pin is only asserted if it is configured to do so for a particular condition;
- 0x0C Current: This read-only command pair returns a signed integer value that is the average current flowing through the sense resistor. It is updated every 1 second with units of 1 mA per bit;
- 0x16 Scaled R: This read-only command pair returns an integer value of the scaled resistance of the cell. It is updated upon a new resistance update in EOS mode only with a resolution of 1 mΩ per bit;
- 0x22 Measured Z: This read-only command pair returns an integer value of the measured impedance of the cell. It is updated upon a new resistance update in EOS mode only with a resolution of 1 mΩ per bit;

- 0x28 InternalTemperature: This read-only command pair returns an unsigned integer value of the internal temperature sensor in units of 0.1°K, measured by the device, and has a range of 0 to 6553.5°K;
- 0x2E StateOfHealth: This read-only command returns an unsigned integer value of the predicted state-of-health (SOH). Where state-of-health is predicted as:

$$SOH = \frac{RemainingAvailableCharge}{DesignCapacity} \cdot 100[\%];$$

- 0x3C DesignCapacity: This read-only command pair returns the expected full charge capacity with units of 1 mAh per bit. The value is stored in Design Capacity.
- 0x3E ManufacturerAccessControl: This read-write word function returns the subcommand that is currently active for reads on MACData(). Word writes to this function will set a subcommand. Commands that do not require data will execute immediately (identical to writes to Control());
- 0x40 MACData: This read-write block returns the result data for the currently active subcommand. It is recommended to start the read at ManufacturerAccessControl() to verify the active subcommand. Writes to this block are used to provide data to a subcommand when required;
- 0x60 MACDataSum: This read-write function returns the checksum of the current subcommand and data block. Writes to this register provide the checksum necessary in order to execute subcommands that require data. The checksum is calculated as the complement of the sum of the ManufacturerAccessControl() and the MACData() bytes. MACDataLen() determines the number of bytes of MACData() that are included in the checksum;
- 0x61 MACDataLen: This read-write function returns the number of bytes of MACData() that are part of the response and included in MACDataSum(). Writes to this register provide the number of bytes in MACData() that should be processed as part of the subcommand. Subcommands that require block data are not executed until MACDataSum() and MACDataLen() are written together as a word;
- 0x79 Cal_Count: Command used during calibration routine for counting the number of samples to average.
- 0x7A Cal_Current: Command used during calibration routine for converting the current measured value in order to evaluate the ccdelta and ccgain;

- 0x7C Cal_Voltage: Command used during calibration routine for converting the voltage measured value in order to evaluate the voltage offset;
- 0x7E Cal_Temperature: Command used during calibration routine for converting the temperature measured value in order to evaluate the temperature offset.

All other Data Commands are used to read a specific parameter that the gauge update itself at regular intervals, depending on the type of data. For example the Current command returns the average current flowing through sensor resistor is updated every one sec when the gauge is in active mode, otherwise the Voltage command returns the value of the measured battery voltage in real time.

Due to the read-only behaviour of this type of commands, in order to be properly executed by the BQ35100 each command needs a transmission followed by a reception on the I²C line.

An example of command sequence for a generic data command is presented in the manual [9] and shown in figure 4.3, where:

- S : represents the START messages;
- P : represents the STOP messages;
- A : represents the ACK messages;
- N : represents the NACK messages.

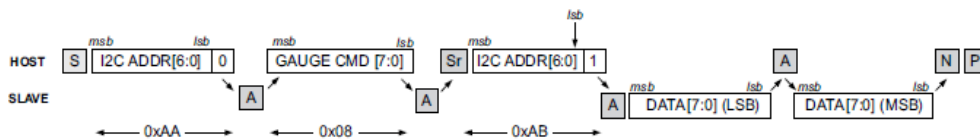


Figure 4.3: Example I²C Data Command Transition Sequence.

4.3.2 Control Subcommands

The Control Subcommands are commands that needs first to send the Control command and after one addition two-command bytes that specifies the particular function desired (TRM [6]). They can be read-only commands or write-only commands, but those that are write-only are commands that instruct the gauge tto do some specific action in certain situations.

The table including all Control Subcommands is referenced in TRM [6] and shown in figure 4.4.

Those commands are responsible for exploiting all the functionalities of the BQ35100

CNTL FUNCTION	CNTL DATA	SEALED ACCESS	DESCRIPTION
CONTROL_STATUS	0x0000	Yes	Reports the status of key features
DEVICE_TYPE	0x0001	Yes	Reports the device type of 0x40 (indicating BQ35100)
FW_VERSION	0x0002	Yes	Reports the firmware version on the device type
HW_VERSION	0x0003	Yes	Reports the hardware version of the device type
STATIC_CHEM_CHKSUM	0x0005	Yes	Calculates chemistry checksum
CHEM_ID	0x0006	Yes	Reports the chemical identifier used by the gas gauge algorithms
PREV_MACWRITE	0x0007	Yes	Returns previous <i>Control()</i> command code
BOARD_OFFSET	0x0009	Yes	Forces the device to measure and store the board offset
CC_OFFSET	0x000A	Yes	Forces the device to measure the internal CC offset
CC_OFFSET_SAVE	0x000B	Yes	Forces the device to store the internal CC offset
DF_VERSION	0x000C	Yes	Reports the data flash version on the device
GAUGE_START	0x0011	Yes	Triggers the device to enter ACTIVE mode
GAUGE_STOP	0x0012	Yes	Triggers the device to stop gauging and complete all outstanding tasks
SELAED	0x0020	No	Places the device in SEALED access mode
CAL_ENABLE	0x002D	No	Toggle CALIBRATION mode enable
LT_ENABLE	0x002E	No	Enables Lifetime Data collection
RESET	0x0041	No	Forces a full reset of the device
EXIT_CAL	0x0080	No	Exit CALIBRATION mode
ENTER_CAL	0x0081	No	Enter CALIBRATION mode
NEW_BATTERY	0xa613	Yes	This is used to refresh the gauge when a new battery is installed and resets all recorded data.

Figure 4.4: Control MAC Subcommands Summary.

as explained below:

- 0x0000 CONTROL_STATUS: This command instructs the device to return status information to Control addresses 0x00/0x01;
- 0x0001 DEVICE_TYPE: When reading DEVICE_TYPE(), a block read is used. This requires that a write to 0x00 of 0x0200 should be followed by a read of 0x40 with 6 bytes to be read out. All In little-endian order, the first 2 bytes are DEVICE_TYPE(), then 2 bytes of FW_VERSION() and 2 bytes of FW_BUILD;
- 0x0002 FW_TYPE: When reading FW_VERSION(), a block read is used. This requires that a write to 0x00 of 0x0200 should be followed by a read of 0x40 with 6 bytes to be read out. All in little-endian order, the first 2 bytes are DEVICE_TYPE(), then 2 bytes of FW_VERSION() and 2 bytes of FW_BUILD;
- 0x0003 HW_TYPE: This command instructs the device to return the hardware version to addresses 0x00/0x01;

- 0x0005 `STATIC_CHEM_CHKSUM`: This command instructs the fuel gauge to calculate chemistry checksum as a 16-bit unsigned integer sum of all static chemistry data;
- 0x0006 `CHEM_ID`: This command instructs the fuel gauge to return the chemical identifier for the programmed chemistry configuration to addresses 0x00/0x01;
- 0x0007 `PREV_MACWRITE`: This command instructs the fuel gauge to return the previous command written to addresses 0x00/0x01. The value returned is limited to less than 0x0020;
- 0x0009 `BOARD_OFFSET`: This command instructs the fuel gauge to calibrate board offset when is in `ACTIVE` mode;
- 0x000A `CC_OFFSET`: This command instructs the fuel gauge to calibrate the coulomb counter offset when in `ACTIVE` mode;
- 0x000B `CC_OFFSET_SAVE`: This command instructs the fuel gauge to save the coulomb counter offset after calibration when it is in `ACTIVE` mode;
- 0x0011 `GAUGE_START`: This command instructs the fuel gauge to enter `ACTIVE` mode;
- 0x0012 `GAUGE_STOP`: This command instructs the fuel gauge to exit `ACTIVE` mode and complete all tasks;
- 0x0020 `SEALED`: This command instructs the fuel gauge to transition from `UNSEALED` state to `SEALED` state;
- 0x002D `CAL_ENABLE`: This command instructs the fuel gauge to enable entry and exit to `CALIBRATION` mode;
- 0x002E `LT_ENABLE`: This command instructs the fuel gauge to enable Lifetime Data collection;
- 0x0041 `RESET`: This command instructs the fuel gauge to perform a full reset. This command is only available when the fuel gauge is `UNSEALED`;
- 0x0080 `EXIT_CAL`: his command instructs the fuel gauge to enable exit to `CALIBRATION` mode;
- 0x0081 `ENTER_CAL`: his command instructs the fuel gauge to enable entry to `CALIBRATION` mode;

- 0xA613 NEW_BATTERY: This command instructs the fuel gauge to prepare itself for the next resistance update and EOS determination to be with a new cell.

An example of command sequence for a generic control subcommand is presented in the SLUA790 manual [9] and shown in figure 4.5.

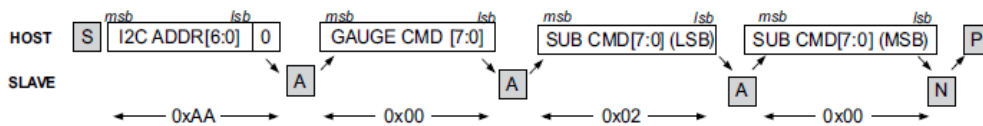


Figure 4.5: Example I²C Control Subcommand Transition Sequence.

4.4 BQ35100 Functions Code

At this point the real code development begins, Where all commands of the BQ35100 are translated in software functions that enable the MCU to command the gauge accordingly to the behaviour wanted.

For this scope a separated library is created with all declarations and implementations of the functions separately to allow the better portability of the software. The developed functions are divided in different sections base on the type of functions they have to exploit. Each functions also contains a detailed description of what the function does and what is needed for.

4.4.1 General Functions

This functions are not directly related to the BQ35100 itself, but they assure to meet specific constrains specified in the manuals.

Transmission and Reception Sequence

The first step in the actual writing of the code is to create a function that allow the MCU to concatenate properly a transmission and a reception reported Listing 4.1.

Listing 4.1: Tx and Rx concatenation function.

```

1 void I2C_Tx_Rx(uint8_t *command, uint16_t commandSize, uint8_t *
  rxData, uint16_t rxSize)
2 {
3     /*Function that is used for commands that, after being sent,
  expects a data in response.
```

```

4      * The parameters used are:
5      * command:      a pointer to an array that represents the command
                      that must be sent to the slave
6      *               divided in bytes from LSB to MSB;
7      * commandSize: the size of the command that must be sent;
8      * rxData:      a pointer to an array that represents the data
                      received from the slave divided in
9      *               bytes from LSB to MSB;
10     * rxSize:      the size of the data that must be received.*/
11     HAL_I2C_Master_Transmit(&hi2c1, BQ_ADDRESS, command, commandSize,
                             50);
12
13     HAL_Delay(1); /* Delay needed in order to satisfy the condition
                      for the delay between
14     * a stop and the successive start that must be at least 66 micro
                      -seconds*/
15     HAL_I2C_Master_Receive(&hi2c1, BQ_ADDRESS, rxData, rxSize, 50);
16 };

```

The function simply uses the HAL commands for I²C transmission and reception and insert between them a delay of 1 *ms* in order to assure the constrain on 66 μ s between a start and a stop condition.

Check Sum Calculation

The CheckSum parameter is a value needed to be sent to the gauge during the sequence to write in the data flash of the gauge and it is reported in List 4.2.

Listing 4.2: Check_Sum Calculation Function.

```

1     uint8_t Check_Sum(uint8_t *value, uint16_t valueSize, uint8_t *memAdd
                      , uint16_t memAddSize)
2     {
3     /*Function that is used for calculating the Check_Sum needed to write
                      correctly in data flash.
4     * The parameters used are:
5     * value:      a pointer to an array that represents the value that
                      must be written to the data memory
6     *               divided in bytes from MSB to LSB;
7     * valueSize:  the size of the value that must be written;
8     * memAdd:     a pointer to an array that represents the memory
                      address divided in bytes from LSB to MSB;
9     * memAddSize: the size of the memory address.
10    * The calculation is performed as followed:
11    * 1) Sum each byte from memAdd and add each byte of the value that
                      must be written;
12    * 2) Truncate to the least significant byte;
13    * 3) Evaluate the complementary of that.*/

```

```

14 uint8_t checksum = 0xFF;
15 uint16_t temporary = 0x0000;
16
17 for(int i = 1; i < memAddSize+1; i++)
18 {
19     temporary += memAdd[i];
20 };
21
22 for(int i = 0; i < valueSize; i++)
23 {
24     temporary += value[i];
25 };
26
27 checksum = 0xFF - (0xFF & temporary);
28
29 return checksum;
30
31 };

```

In practice the command takes the command that must be written into memory and the address of every location that must be written, each byte is added with the others and it takes the complementary of the result. The procedure is accomplished as reported in TRM [6].

Data Length Calculation

As the previous one this command is related to the operation of write on the data flash of the gauge and it is reported in Listing 4.5.

Listing 4.3: Data Length Calculation function.

```

1 uint8_t Data_Len(uint16_t valueSize)
2 {
3     /*Function that is used for calculating the Length needed to write
4     correctly in data flash.*/
5     uint8_t length = 0x00;
6
7     length = 0x04 + valueSize;
8
9     return length;
10 };

```

The function calculates the length needed to write correctly in the data flash, by adding four to the size of the value that must be written.

Floating Point Conversion

The function is use during the current calibration and it is needed to shift the values of CC_Gain and CC_Delta from a integer value to a floating point representation divided in sign, mantissa, exponent and base. Those values must be evaluated properly and stored in memory to have correct current measurements.

The TRM page 18 reports a flow chart of the rational flow to proper convert the values into some that can be correctly be written in data flash[6]. (The size of the figure does not allow it to be reported here.

Listing 4.4: Floating Point Conversion function.

```

1 void Float_Conversion(float value, uint8_t *rawData)
2 {
3 /* Function that convert a value from floating point representation
4  in a suitable
5  * representation divided by bytes.*/
6  float val, module, tmp_val, k = 256.0;
7  int exp = 0;
8  uint8_t byte[3];
9
10 val = value;
11 if (val < 0)
12 {
13     module = -val;
14 }else
15 {
16     module = val;
17 }
18 tmp_val = module;
19 if (tmp_val < 0)
20 {
21     while (tmp_val < 0)
22     {
23         tmp_val = 2 * tmp_val;
24         exp++;
25     }
26 }
27 if (exp > 127)
28 {
29     exp = 127;
30 }else if (exp < -128)
31 {
32     exp = -128;
33 }
34 for (int i = 0; i < exp ;i++)
35 {
36     k = k/(float)2;

```



```

36     }
37     tmp_val = k * module - 128;
38     byte[2] = (uint8_t)tmp_val;
39     tmp_val = 256 * module - byte[2];
40     byte[1] = (uint8_t)tmp_val;
41     tmp_val = 256 * module - byte[1];
42     byte[0] = (uint8_t)tmp_val;
43     if (val < 0)
44     {
45         byte[2] = byte[2] | 0x80;
46     }
47     rawData[0] = (uint8_t)(exp + 128);
48     rawData[1] = byte[2];
49     rawData[2] = byte[1];
50     rawData[3] = byte[0];
51 };

```

Creation of Command Stream for Memory Commands

The function takes an address of the data memory that is a `uint16_t` data and converts it in an array of `uint8_t`.

Listing 4.5: Command Stream Creation function.

```

1 void From_Addr_To_Command( uint8_t *command, uint16_t memAddress)
2 {
3 /* This functions is used to convert the address memory that is
4  * targeted for the
5  * operation and transform it in a command suitable for I2C standard.
6  * The parameters are:
7  * command      = the array that contains the command to be sent in
8  *               I2C format ,
9  *               the dimension starts from 1 because the access in
10 *               memory needs
11 *               primary a MAC command and than the memory address;
12 * memAddress   = the address of the cell memory targeted for the
13 *               operation.*/
14     command[2] = memAddress >> 8;
15     command[1] = memAddress & 0xff;
16 };

```

4.4.2 Data Commands

In this section the structure of a data command type.
All data commands uses the same structure:

1. writing the command to the gauge,
2. reading from the gauge as many bytes as required accordingly to the command send.

An example of a data command structure is shown in Listing 4.6, where the Accumulated_Capacity command is shown.

Listing 4.6: Accumulated Capacity Function.

```

1 uint32_t BQI2C_Accumulated_Capacity ()
2 {
3 /* Function that reads four bytes the value in uAh of the accumulated
4 * coulombs since the coulomb counter was started. If it reaches the
5 * maximum value it will hold the full count. */
6
7     uint32_t acc_cap;
8     uint8_t buffer [4];
9
10    I2C_Tx_Rx(&ACC_CAP, 1, buffer , 4);
11    acc_cap = (buffer [3] << 24) | (buffer [3] << 16) | (buffer [1] << 8) |
12    buffer [0];
13
14    return acc_cap;
15 };

```

First the MCU sends the command (code 0x02), after that four bytes are read and finally the bytes are sorted together to make it possible for the MCU to read the result correctly.

4.4.3 Command MAC Subcommands

As said previous in section 4.3.2 the command MAC subcommands are commands that enables to exploit all functionalities of the BQ35100.

The structure of this type of command is the same for each one, so all of them shares the same construction.

As example of a subcommand the New_Battery() is shown in Listing 4.7.

Listing 4.7: New_Battery Functions.

```

1 void BQI2C_New_Battery ()
2 {
3 /* This function instructs the gauge to update for the next
4 * resistance and EOS
5 * determination with a new cell.*/
6     uint8_t new [3];

```

```

7 |     new[0] = CONTROL;
8 |     From_Addr_To_Command(new, NEW_BATTERY);
9 |     HAL_I2C_Master_Transmit(&hi2c1, BQ_ADDRESS, new, 3, 50);
10 |     HAL_Delay(1);
11 | };

```

As shown the command MAC subcommands follow a specific procedure, as written in the TRM [6]:

1. send the Control command first (code 0x00),
2. send the LSB part of the command (code 0x13) ,
3. send the MSB part of the command (code 0xA16).

This procedure ensures the correct interpretation of the command by the gauge. Although all this type of commands share the same procedure there are some that requires additional steps because they may fail to be received or that requires some time when received to proper trigger a bit in Control Status Register. So in order to properly handle those specific situation the command itself provides a check on the proper bit until it is settled. An example the Exit_Cal() is in Listing 4.8.

Listing 4.8: Exit_Calibration Functions.

```

1 | void BQI2C_Exit_Calibration()
2 | {
3 |     /* When in active mode this function instructs the gauge to exit
4 |        calibration mode.
5 |        * It also contains the routine that controls if the command is
6 |        received and
7 |        * performed by the gauge. After it is correctly performed a
8 |        BQI2C_Gauge_Stop()
9 |        * is used to exit active mode.*/
10 |
11 |     int cal_ex_flag = 0;
12 |     uint8_t cal[3];
13 |     uint16_t cnt_status;
14 |
15 |     cal[0] = MAC;
16 |     From_Addr_To_Command(cal, EXIT_CAL);
17 |     HAL_Delay(1);
18 |
19 |     do{
20 |         HAL_I2C_Master_Transmit(&hi2c1, BQ_ADDRESS, cal, 3, 50);
21 |         HAL_Delay(50);
22 |         cnt_status = BQI2C_Control_Status();
23 |         if((cnt_status | 0xEFFF)== 0xEFFF)
24 |         {
25 |             cal_ex_flag = 1;
26 |         }
27 |     };

```

```

23     HAL_Delay(500);
24     } while (cal_ex_flag != 1);
25
26     BQI2C_Gauge_Stop();
27     do{
28         cnt_status = BQI2C_Control_Status();
29         HAL_Delay(500);
30     } while (!(cnt_status & G_DONE_MASK));
31 };

```

As shown a continuous check on the Cal_Mode bit on Control_Status to control if the gauge actually exits the calibration mode or not.

And at the end of it the function controls that the G_DONE byte is set, which means that the gauge completed its task and can be powered down.

4.4.4 Memory Commands

In this section the commands listed are those that implement the operations relative to the data memory (reading and writing).

In order to be able to interact with the data memory a specific sequence of command must be followed that differs a little for reading and writing, as described in the gauge communication manual [10].

For reading from memory:

1. send MAC command,
2. send MAC_Data command,
3. read as many bytes as needed.

The function shown in Listing 4.9 is the one exploiting the sequence of commands to read a certain address of the memory.

Listing 4.9: Read Memory Functions.

```

1 void BQI2C_Read_Mem(uint16_t mem_Address, uint16_t memAdd_Size,
2     uint8_t *buffer)
3 {
4     /* This functions is used to read from the data flash of the gauge.
5     * The values inside data flash are divided in bytes, so if one data
6     * is larger than a byte a consecutive reading is performed.
7     * The parameters used are:
8     * mem_Address = the address of the first-cell of the wanted value
9     *               in the data flash;
10    * memAdd_Size = the dimension of the data stored that must be read;

```

```

10 * buffer      = a pointer to an array of memAdd_Size dimension
    needed
11 *
    to save the value of the data divided in bytes from
12 *           the MSB to the LSB*/
13 uint8_t comando[3] = {};
14
15 From_Addr_To_Command(mem_Address, comando);
16 comando[0] = MAC;
17
18 HAL_I2C_Master_Transmit(&hi2c1, BQ_ADDRESS, comando, 3, 50);
19 HAL_Delay(1);
20 HAL_I2C_Master_Transmit(&hi2c1, BQ_ADDRESS, &MAC_DATA, 1, 50);
21 HAL_Delay(1);
22 HAL_I2C_Master_Receive(&hi2c1, BQ_ADDRESS, buffer, memAdd_Size,
    50);
23 HAL_Delay(1);
24 };

```

The function is used to read a memory cell from the memory and requires the address of the first memory to be read and the size of the data that must be read and returns the value divided per bytes.

For writing into memory the procedure is more complex:

1. calculate Check_Sum and Data_Len,
2. send MAC command,
3. send MAC_Data followed by the address of the memory (LSB first),
4. send Check_Sum,
5. send Data_Length.

The function shown in Listing 4.10 is the one exploiting the sequence of commands to write into the memory.

Listing 4.10: Write Memory Functions.

```

1 void BQI2C_Write_Mem(uint16_t mem_Address, uint16_t memAdd_Size,
    uint8_t *data)
2 {
3 /* This functions is used to write a data to the data flash of the
    gauge.
4 * The values inside data flash are divided in bytes, so if one data
5 * is larger than a byte a consecutive writing is performed.
6 * The parameters used are:
7 * mem_Address = the address of the first-cell that must be written
8 *           in the data flash;

```

```

9  * memAdd_Size = the dimension of the data stored that must be
    written;
10 * data      = a pointer to an array of memAdd_Size dimension that must
11 *           be written divided in bytes from the MSB to the LSB
    */
12 uint8_t comando[3] = {};
13 uint8_t M_D[memAdd_Size + 1];
14 uint8_t M_C[2] = {MAC_DATA_SUM, 0x00};
15 uint8_t M_L[2] = {MAC_DATA_LEN, 0x00};
16
17
18 From_Addr_To_Command(comando, mem_Address);
19 comando[0] = MAC;
20 M_D[0] = MAC_DATA;
21 for (int i = 1; i < memAdd_Size+1; i++)
22 {
23     M_D[i] = data[i-1];
24 };
25 M_C[1] = Check_Sum(data, memAdd_Size, comando, 2);
26 M_L[1] = Data_Len(memAdd_Size);
27
28 HAL_I2C_Master_Transmit(&hi2c1, BQ_ADDRESS, comando, 3, 50);
29 HAL_Delay(1);
30 HAL_I2C_Master_Transmit(&hi2c1, BQ_ADDRESS, M_D, memAdd_Size + 1,
    50);
31 HAL_Delay(1);
32 HAL_I2C_Master_Transmit(&hi2c1, BQ_ADDRESS, M_C, 2, 50);
33 HAL_Delay(1);
34 HAL_I2C_Master_Transmit(&hi2c1, BQ_ADDRESS, M_L, 2, 50);
35 HAL_Delay(1);
36 };

```

4.5 Data Memory Mapping

In addition to the product code, in order to make the code adaptable to any possible usage with all kind of battery having different chemistry or for different scopes, a code that maps all data memory cells of the BQ35100 is made.

The characteristics of a single cell in the data memory that differs one another are:

1. Cell Address: that is an hexadecimal value from 0x4000 to 0x43FF;
2. Cell Size; that can be one, two, four or eight.

For this reason to better handling the mapping of the memory of the BQ35100, a struct, presented in Listing 4.11, is used to describe each memory cell.

Listing 4.11: Example of The Struct used to Map a Cell From Data Memory.

```

1 typedef struct
2 {
3     const uint16_t Mem_Address;           /*Code of Memory Address*/
4     const uint16_t Mem_Cell_Dimension;   /*Dimension of the data*/
5 }Memory_Cell;

```

Indeed the code is very self explicative, it describes a generic memory cell by using the address and the dimension of the cell accordingly to the TRM [6].

Depending of the data that is taken into account, it must be consider that inside of the memory there are unsigned, integers, floating point values and also a string. So when reading/writing into a specific memory cell it must be taken into account. An example of the code made for mapping the data memory is shown in Listing 4.12, where *Op_Config_A* and *OT_Dsg* cell are described.

Listing 4.12: Example of Memory Cells Description.

```

1 Memory_Cell Operation_Config_A = {0x41B1, ONE};
2 /*Operation Config A: used to change setting parameters of
3  * the gauge accordingly to the needs*/
4
5 Memory_Cell OT_Dsg = {0x41D6, TWO};
6 /*Maximum Temperature Threeshold: when the temperature,
7  * measured by internal sensor or the external NTC,
8  * reaches the setted level for OT_Dsg_Time seconds the
9  * ALERT signal is triggered (if the corrisponding alert
10 * pin was enabled in Alert_Config cell)*/

```

Each memory cell keeps the same name as the one reported in TRM and also it is provided with a simple description of the data that is held inside.

Chapter 5

Firmware Test

The aim of this chapter is to underline the reasons behind the test procedure developed to test the produced firmware during a complete discharge of a full capacity battery.

5.1 Previous Consideration for Test Procedure

Before even take into account the test project there are many considerations that must be done before:

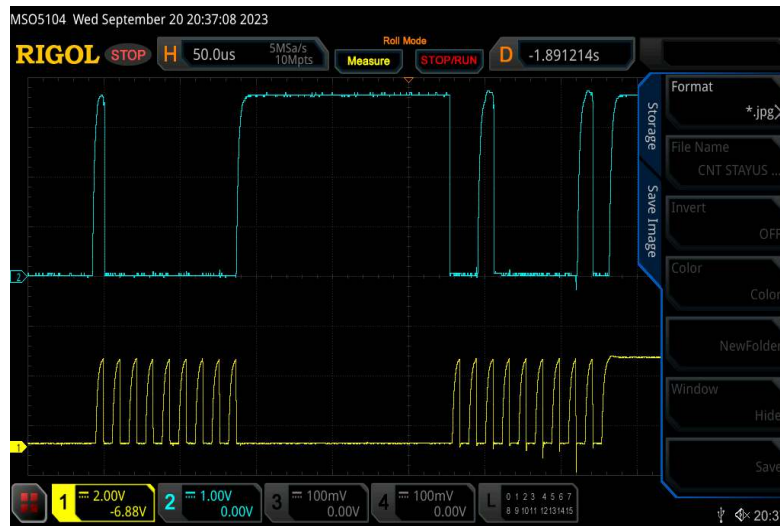
1. the current consumption of a real case scenario of an electronic system for precision agriculture;
2. the amount of time the test will take;
3. the condition in which the test can be made;
4. the expected battery life of the battery;
5. the data of the BQ35100 that are meaningful for the analysis;
6. possible problematic that can be encountered during a real case scenario.

So the best choice is to firstly test individually the commands produced and after that to design a proper stress test of the battery, in accordance to the manuals in order to try to achieve the best possible results.

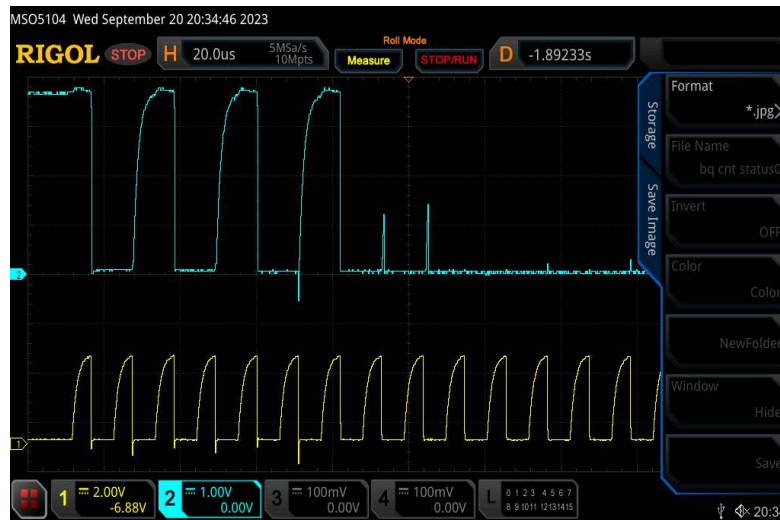
5.1.1 Individual Command Test

The first step is the individual test of each command in order to see if they work properly and also to see how the BQ35100 responds to some commands that change its status in run-time.

As example the command `Control_Status` and the reception of the data is shown in figure 5.1.



(a) Command Transmission from MCU to the gauge



(b) Data Reception from the gauge to MCU

Figure 5.1: Control_Status Command Example

The figure shows the SCL signal in the channel 1 (yellow one) and the SDA

signal in channel 2(blue one) in the oscilloscope. The first figure represents the first part of the procedure where the BQ address and the Control_Status command are sent (0xAA for writing and 0x0000), the second figure shows the final reading part of the procedure where two bytes are read from the BQ (0x80 and 0x20) LSB first.

5.2 Current Consumption Estimation

After the evaluation of the commands accuracy, the current consumption of the system in a real case scenario must be evaluated in order to estimate the battery discharge time.

The consumption regarding a real case scenario of a system for smart agriculture using the LS14500 are taken from two different studies one that provides the current consumption measurements of a ultra low power, wireless, battery-powered node for measuring soil water content [11] and the second study that provides the consumption of an STM32 MCU battery-powered node to manage the control stage of a drip irrigation system [12].

The figures below shows the current consumption monitored in the two case of study.

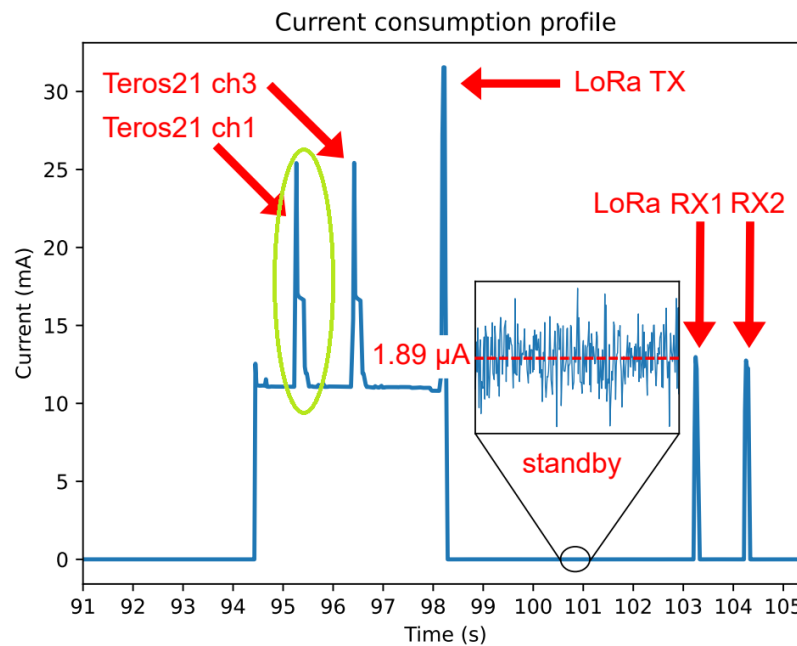


Figure 5.2: Consumption Monitored From System Monitoring Soil Water Content [11].

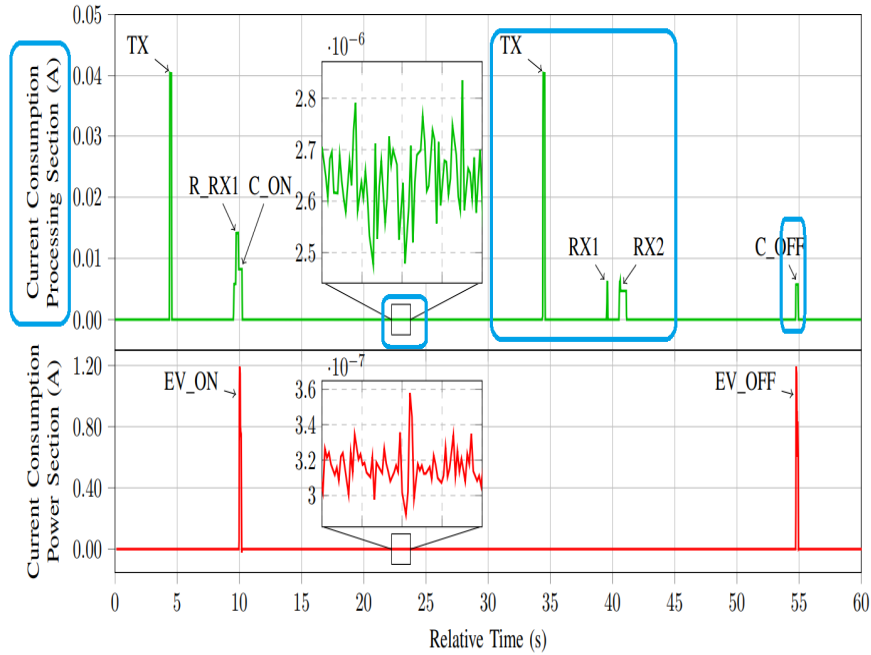


Figure 5.3: Consumption Monitored From Drip Irrigation System [12].

In particular:

- in the figure 5.2 the consumption of the Teros21 sensor (enlightened in green circle) is the one that is important for the considerations made;
- in the figure 5.3 the control stage consumption curve (green curve) is the one of interest and, more specifically, the second cycle of LoRa-node sending frame and the subsequent ignition of the MCU is taken into account (circled in blue).

5.2.1 Python Code For Current Consumption

In order to analyze the data discussed before a python code is written in order to evaluate the average current consumption of the STM32 MCU (`current_Lora` in the code) and the current consumption of the Teros21 sensor when activated (`current_Sensor_eff` in the code) and to estimate the battery end-of-life time in both real case scenario and a possible test. The code part for the current consumption calculation is shown in Listing 5.1, where a trapezoidal approximation among intervals of $20ms$ over each of the experimental results is used. After that the average current over one hour is calculated.

Listing 5.1: Python Code For Current Consumption.

```

1 import os
2 from scipy.integrate import trapz
3
4 def read_file(SpringF, times, readings, file_name): # checked
5     file = open(file_name, 'r')
6     sleep_value = 0.000026
7     previous_value = 0.0
8     for line in file:
9         parameters = line.split(',')
10        value = max(float(parameters[0]), 0.000018)
11        if "sensor" in file_name:
12            runtime_current = 0.01107 - 0.005741
13            t0 = 95.22
14            if (float(parameters[1]) > 95.22) and (float(parameters
15 [1]) < 96.36):
16                readings.append(max(value - runtime_current,
17 sleep_value))
18                times.append(float(parameters[1]) - t0)
19            else:
20                dt = 0.0
21                if SpringF == 9:
22                    dt = 0.3492
23                    t0 = 34.35
24
25                if (float(parameters[1]) > 34.35) and (float(parameters
26 [1]) < 41.14):
27
28                    if float(parameters[1]) < 34.35 + dt:
29                        value = max(value, previous_value)
30                        previous_value = value
31                        times.append(float(parameters[1]) - t0)
32                        readings.append(value)
33
34        file.close()
35    ...
36
37 if __name__ == '__main__':
38    ...
39    SF_12_t_c = []
40    SF_9_t_c = []
41    SF_12_v_c = []
42    SF_9_v_c = []
43    SF_12_t_s = []
44    SF_9_t_s = []
45    SF_12_v_s = []
46    SF_9_v_s = []
47    ...

```

```

46 pw_LORA = trapz(SF_9_v_c, SF_9_t_c)
47 pw_Sensor = trapz(SF_9_v_s, SF_9_t_s)
48 current_Lora = 1000 * pw_LORA / delta
49 current_Sensor_eff = 6 * 1000 * pw_Sensor / delta
50 ...
51

```

The code is designed to take into account only the signal parts circled in figure 5.2 and figure 5.3. For the consumption of the soil sensor the system can have six of them activated at the same time so the consumption of six of them is taken as worst case scenario. Then the average consumption is calculated among the period of each transmission (for $SF = 9$ is 35 seconds). The results of the average current consumption are the following:

$$I_{Lora} = 0.00441 \text{ mA} \quad I_{Sensor} = 0.01315 \text{ mA} \quad I_{Sleep_Mode} = 0.0026 \text{ mA}$$

5.2.2 Estimation of Time to End

With the results obtained above the battery time-to-end can be estimated, by considering also the average current consumption of the BQ35100 in EOS mode.

According to manual [8] and manual [13], the cycle for a proper EOS updating data must follow some conditions:

1. if the interval between different measurements is less than 1 minute than the gauge must be not powered down,
2. the Gauge_Start command must be received prior any major discharge,
3. after Gauge_Stop command is received 15 seconds must be wait prior to any other action in order to allow the gauge algorithm to perform the internal resistance data updating.

The part of the code in Listing 5.2 evaluates the average consumption of the gauge in EOS mode over a time period of one hour.

Listing 5.2: Python Code for BQ35100 EOS Mode Consumption.

```

1 ...
2 pw_BQ = (0.315 * 14 + 0.075 * 15 + 0.00005 * 3571)
3 current_BQ = pw_BQ / delta #mA
4 ...

```

The average current consumption results:

$$I_{BQ_EOS} = 0.001587 \text{ mA}$$

Now the life time estimation of the battery can be made by using the formulas below:

$$CAP = Battery_Capacity = 2600 mAh$$

$$I_{average} = I_{Lora} + I_{Sensor} + I_{Sleep_Mode} + I_{BQ_EOS} = 0.03714mA$$

$$Expected_Time_to_End[days] = \frac{CAP}{I_{average} * 24[hour/day]} \approx 2917days$$

The same evaluation is made also by using a python code shown in Listing 5.3

Listing 5.3: Python Code Time to End Estimation in Norml Conditions.

```

1 ...
2 # average current consumption in normal conditions
3 I_normal_condition = current_BQ + current_Sensor_eff +
4   current_Sleep + current_Lora # mA
5 # number of days for discharge in normal conditions
6 days_for_discharge_normal_condition = battery_max_capacity / (
7   I_normal_condition * hours_for_day) # days
8 print("Giorni effettivi: " + str(
9   days_for_discharge_normal_condition))
10 ...

```

It becomes clear that a test in real case consumption conditions would require to much time to discharge the battery, so more effective stress is needed.

5.3 Possible Procedures

Taking into accounts all said before, it is now necessary to understand how to speed up a test in such a way that the BQ35100 can predict properly the battery status. By looking at the TRM manual, it is clear that can be two different approaches for a testing procedure: one that requires the gauge to be always on to measure every single discharge from the system, the other procedure requires different a different pulse to force a little discharge and to enable the gauge to make measurements of the parameters needed for the gauge algorithm to evaluate the battery status.

5.3.1 First Test Procedure

The first stress procedure is based on the prospect to make it last for a maximum of 14 days.

After the time limit set than the average current consumption of the system:

$$\bar{I} = \frac{BAT_Capacity}{14days * 24\frac{hour}{days}} \approx 7.74mA$$

As discussed section 5.2.1 the average consumption of the MCU and the TEROS21 sensor doesn't allow a system to speed up.

To make this possible instead of using multiple sensor to induct a higher current consumption to speed up the procedure, the best choice is to use a resistor as a load for a certain amount of time. Some calculation are made in order to consider a proper resistor to use, taking into account that when $SF = 9$ the LoRa protocol needs at least 35 seconds before another possible transmission.

Calculation made are shown below:

$$\bar{I} = \bar{I}_{LoRa} + \bar{I}_{Load} + \bar{I}_{BQ} + \bar{I}_{Sleep}$$

So considering to have a constant consumption on the resistance for 7 seconds:

$$\bar{I}_{Load} \approx 7.73mA \longrightarrow I_{Load} \approx 38.65mA = \frac{BAT_Voltage}{R_{Load}} \longrightarrow R_{Load} \approx 82\Omega$$

Because in this case the data must be collected for each LoRa cycle, it becomes clear that switching on and off the gauge can only slow down the test so for these reason the BQ35100 will be always kept on by the MCU.

But even if it is said that it is possible in the TRM, this can make the gauge reach an unkwown state and the results will show how this particular usage will affect the detection.

So in this case the estimated duration of the test will be:

$$Days = \frac{BAT_Capacity}{\bar{I} * 24 \frac{hour}{days}} = 13.5days.$$

5.3.2 Second Test Procedure

The second test procedure is thought on two different assumptions:

- LiSOCl₂ batteries needs a proper amount of time to rest after a major discharge,
- the internal resistance evaluation made by the BQ35100 for predicting EOS condition are not time-related so it only requires a short pulses to take measurements and do not require to be active on every discharge event.

This being said, the procedure will has four different phases:

1. LoRa Transmission and Reception windows;
2. After seven seconds the gauge is activated a little pulse is uses to make measurements and after that the gauge is disabled;
3. One hour of a very high current consumption using the same resistance as before;

4. One hour of letting the battery rest.

In this condition the expected duration for the entire testing procedure will be approximately 4 days.

5.4 Final Setup

For both testing procedures the BQ35100-EVM is used [14]. The functional block scheme with the set-up for the test is shown in figure 5.4.

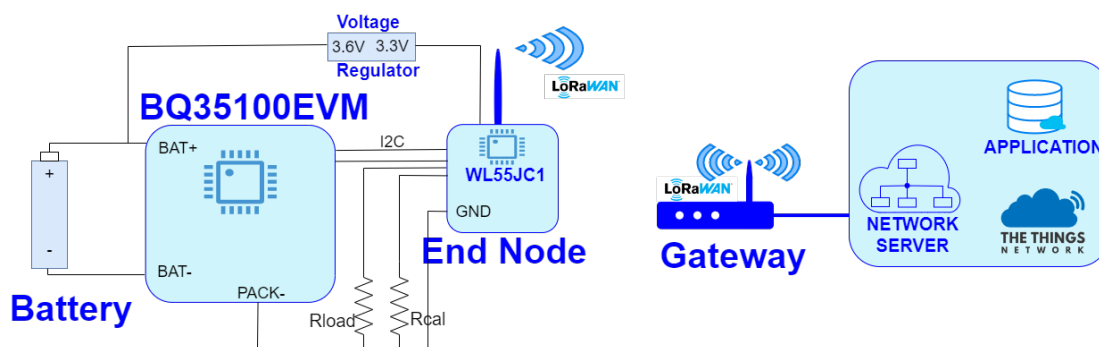


Figure 5.4: Setup Configuration for Testing Procedure.

After the test is launched the data will be send by the MCU to the LoRa gateway where they will be available to read.

The flow chart with the procedures flow is shown in figure 5.5.

5.5 Test Code

Taking into account the previous consideration a dedicated code for the testing procedures is made.

In each section there will be both codes produced so when the first one is explained and for the second one only the differences with the previous one will be highlighted.

5.5.1 Starting project

The firmware made for the test uses the library explained in chapter 4 in combination with the firmware dedicated to handle the LoRaWAN protocol, starting from an example project for the board under exam LoRaWAN_End_Node (Source [15]) for STM32-WL55JC1 that is the same for both testing procedures.

The project implements initialization of all needed peripherals, in particular it takes

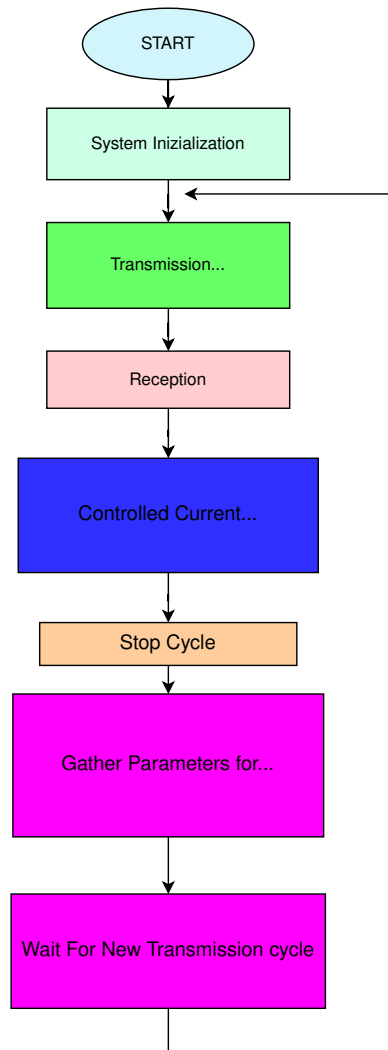


Figure 5.5: Flow Chart of Testing Procedures.

care of the initialization the LoRa protocol and the handling of the communication for Class-A end device like shown in figure 1.3.

The following section shows the the pieces of code added to the original one in order to adapt it to the test procedure for the detection of the battery end-of-life condition.

5.5.2 Main function

The Listing 5.4 reports the main function with initialization of all the peripherals uses in the first procedure.

Listing 5.4: main.

```

1 int main(void)
2 {
3     /* USER CODE BEGIN 1 */
4     uint16_t c_s = 0x0000;
5     /* USER CODE END 1 */
6     /* MCU Configuration
7     _____*/
8     /* Reset of all peripherals, Initializes the Flash interface and
9     the SysTick. */
10    HAL_Init();
11    ...
12    /* Configure the system clock */
13    SystemClock_Config();
14    ...
15    /* Initialize all configured peripherals */
16    ...
17    /* USER CODE BEGIN 2 */
18    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
19    /*Wait for BQ initialization achieve.*/
20    HAL_Delay(500);
21    do{ /*Control if initialization is achieved*/
22        c_s = BQI2C_Control_Status();
23        HAL_Delay(500);
24    }while (!(c_s & INIT_COMP_MASK));
25    /* USER CODE END 2 */
26    /* Infinite loop */
27    /* USER CODE BEGIN WHILE */
28    while (1)
29    {
30        /* USER CODE END WHILE */
31        MX_LoRaWAN_Process();
32        /* USER CODE BEGIN 3 */
33    }
34    /* USER CODE END 3 */
35 }

```

The addition made is the setting of the GPIO_PIN_5 (directly connected with the GE pin of the BQ35100) and while for controlling the accomplishment of the internal initialization of the BQ35100 by reading if the INITCOMP pin of the Control_Status register is asserted.

And after the condition is verified then the code moves to the while loop.

For the second procedure there is no change in the main function because the gauge does not need to be always on.

5.5.3 LoRaWAN Protocol Functions

In this section the additions to the LoRa protocol handling functions are implemented.

Initialization of LoRa Stack

- **First Procedure:**

The addition to the initialization function of the LoRa protocol is shown in 5.5.

Listing 5.5: LoRaWAN_Init For Procedure 1.

```

1 void LoRaWAN_Init(void)
2 {
3   ...
4   /* USER CODE BEGIN LoRaWAN_Init_Last */
5   UTIL_TIMER_Create(&SxTimer, 7000, UTIL_TIMER_ONESHOT,
6   OnSxTimerEvent, NULL);
7   UTIL_TIMER_Create(&SxEndTimer, 7000, UTIL_TIMER_ONESHOT,
8   OnSxEndEvent, NULL);
9   UTIL_TIMER_Create(&DataTimer, 13000, UTIL_TIMER_ONESHOT,
10  OnDataTimer, NULL);
11  /* USER CODE END LoRaWAN_Init_Last */
12  ...
13 }

```

The code creates three different timers that are needed to handling the EOS routine.

In specific:

1. **SxTimer:** this timer is used to introduce a delay before the burst current consumption on the resistor to give time to complete the LoRa cycle, the reload value can be both set to 1s and to 7s depending on if a downlink message is received or not;
2. **SxEndTimer:** this timer is used to activate the burst current consumption;
3. **DataTimer:** this timer is used to introduce a delay to let the proper amount of time for the gauge algorithm to evaluate the battery status, accordingly to the TRM [6].

- **Second Procedure:**

The addition to the initialization function of the LoRa protocol is shown in 5.6.

Listing 5.6: LoRaWAN_Init For Procedure 2.

```

1 void LoRaWAN_Init(void)
2 {
3   ...
4   /* USER CODE BEGIN LoRaWAN_Init_Last */
5   UTIL_TIMER_Create(&MeasurementTimer, 7000, UTIL_TIMER_ONESHOT,
6     MeasurementTimer, NULL);
7   UTIL_TIMER_Create(&BurstModeTimer, 3600000, UTIL_TIMER_ONESHOT,
8     EndBurstEvent, NULL);
9   UTIL_TIMER_Create(&RestModeTimer, 3600000, UTIL_TIMER_ONESHOT,
10    EndRestEvent, NULL);
11  /* USER CODE END LoRaWAN_Init_Last */
12  ...
13 }

```

As the previous one this code creates three different timers that are needed to handling the EOS routine.

In specific:

1. **MeasurementTimer**: this timer is used to introduce a delay before the burst current consumption on the resistor to give time to complete the LoRa cycle;
2. **BurstModeTimer**: this timer is used to activate the burst current consumption;
3. **RestModeTimer**: this timer is used to let the battery rest.

LoRa Preparation of Transmission Frame

The function SendTxData prepares uploads the data in AppDataBuffer before sending the buffer in the next transmission session. The common addition for both procedures on the code is in Listing 5.7.

Listing 5.7: SendTxData.

```

1 static void SendTxData(void)
2 {
3   ...
4   if (LmHandlerIsBusy() == false)
5   {
6     ...
7     AppData.Port = LORAWAN_USER_APP_PORT;
8     ...
9     if ((LmHandlerParams.ActiveRegion == LORAMAC_REGION_US915) || (
10      LmHandlerParams.ActiveRegion == LORAMAC_REGION_AU915)
11      || (LmHandlerParams.ActiveRegion == LORAMAC_REGION_AS923))
12     ...

```

```

12  else
13  {
14      /* Updating Payload for Uplink*/
15      for(i = 0; i < 19; i++)
16      {
17          AppData.Buffer[i] = up_payload[i];
18      }
19      for( i = 19; i < 24; i++)
20      {
21          AppData.Buffer[i] = 0x00;
22      }
23  }
24  AppData.BufferSize = i;
25  ...
26  status = LmHandlerSend(&AppData, LmHandlerParams.IsTxConfirmed,
27  false);
28  ...
29  }
30  ...
31  }

```

The code added loads the data created in the AppDataBuffer: the first twenty bytes are from actual data read from the gauge, the other twenty-five bytes are used to simulate a real time scenario data frame in which forty four bytes are needed. The differences in the procedures begins in line 29 of the code.

- **First Procedure:**

in the first procedure after preparing the payload to be send and the transmission of the data then the timer for a new transmission is started again as shown in Listing 5.8;

Listing 5.8: SendTxData Proedure 1.

```

1  ...
2  if (EventType == TX_ON_TIMER)
3  {
4      UTIL_TIMER_Stop(&TxTimer);
5      UTIL_TIMER_SetPeriod(&TxTimer, MAX(nextTxIn, TxPeriodicity));
6      UTIL_TIMER_Start(&TxTimer);
7  }
8  ...
9

```

- **Second Procedure:**

The code control if the join between the LoRa node and the LoRa gateway is established, by controlling the join flag, and if it is so then the MeasurementTimer is started, otherwise the transmission timer is started so another

transmission is needed. This was made in order to avoid missing measurements due to burst consumption before the join between the end node and the gateway is established, as shown in Listing 5.9.

Listing 5.9: SendTxData Procedure 2.

```

1  ...
2  if (EventType == TX_ON_TIMER)
3  {
4      if (join == 1)
5      {
6          UTIL_TIMER_Stop(&MeasurementTimer);
7          UTIL_TIMER_Start(&MeasurementTimer);
8
9      } else
10     {
11         UTIL_TIMER_Stop(&TxTimer);
12         UTIL_TIMER_SetPeriod(&TxTimer, MAX(nextTxIn,
TxPeriodicity));
13         UTIL_TIMER_Start(&TxTimer);
14     }
15 }
16 ...
17

```

LoRa Transmission Timer Callback

The OnTxTimerEvent callback is called when the TxTimer reaches the reload value. The callback is shown in Listing 5.10.

Listing 5.10: OnTxTimerEvent.

```

1  static void OnTxTimerEvent(void *context)
2  {
3      /* USER CODE BEGIN OnTxTimerEvent_1 */
4      MX_I2C1_Init();
5      int exit_loops = 0;
6      if (down_payload[0] == 0x10)
7      {
8          /*Change Max Temperature Command*/
9          uint16_t user_Temperature = 0x0000;
10         user_Temperature = (down_payload[0] << 8 | down_payload[1]);
11         BQI2C_Write_OT_Dsg(user_Temperature);
12
13     } else if ((cal_done == 1) && (down_payload[0] == 0x00))
14     {
15         BQI2C_Calibration_Routine();
16         cal_done = 0;
17     } else if ((new_BAT_flag == 1) && (down_payload[0] == 0x01))

```

```

18 {
19     BQI2C_New_Battery ();
20     BQI2C_From_Sealed_to_Unsealed ();
21     BQI2C_From_Unsealed_to_FullAccess ();
22     up_payload [15] = 0x00;
23     down_payload [0] = 0xFF;
24     new_BAT_flag = 0;
25 }
26 ...
27 }
28 }

```

When the callback is called a control on the the received command (`down_payload`) is done and the proper action is taken accordingly.

Based on the received sequence of bytes:

- if the first command byte received is **0x10** then the temperature value inserted by the user is converted into a single data of two bytes and write it in the data memory of the gauge;
- if the **0x01** then the new battery command is received by the user so it means that a new battery is inserted and the MCU firstly send the `New_Battery` command;
- if the command received is **0x00** than the calibration routine is activated and the MCU uses another resistor connected to another pin in order to calibrate the gauge Voltage, Current and Temperature.

The differences in the procedures begins in line 27 of the code.

- **First Procedure:**

in the first procedure after controlling the received command from the host, the periodicity of the `SxTimer` is set to 7s and the gauge is activated by send the `Gauge_Start` command and a control if the gauge switches to activation mode. After all this the transmission timer and `SxTimer` are started; as shown in Listing 5.11;

Listing 5.11: `OnTxTimerEvent` Procedure 1.

```

1     ...
2     UTIL_TIMER_SetPeriod(&SxTimer , 7000);
3     BQI2C_Gauge_Start ();
4     do{
5         HAL_Delay(500);
6         control_status = BQI2C_Control_Status ();
7         exit_loops += 1;

```

```

8   }while (!(control_status & GAUGE_ACTIVE_MASK) && (exit_loops <
9   100));
9   if (exit_loops > 100)
10  {
11      Reset = 1;
12  }
13  ...
14      /*Wait for next tx slot*/
15  UTIL_TIMER_Start(&TxTimer);
16  /* USER CODE BEGIN OnTxTimerEvent_2 */
17  UTIL_TIMER_Start(&SxTimer);
18  /* USER CODE END OnTxTimerEvent_2 */
19  ...
20

```

- **Second Procedure:**

The code control if the join between the LoRa node and the LoRa gateway is established, by controlling the join flag, and if it is so then the MeasurementTimer is started, otherwise the transmission timer is started so another transmission is needed.

This was made in order to avoid missing measurements due to burst consumption before the join between the end node and the gateway is established, as shown in Listing 5.12.

Listing 5.12: OnTxTimerEvent Procedure 2.

```

1   ...
2   if (EventType == TX_ON_TIMER)
3   {
4       if (join == 1)
5       {
6           UTIL_TIMER_Stop(&MeasurementTimer);
7           UTIL_TIMER_Start(&MeasurementTimer);
8
9       } else
10      {
11          UTIL_TIMER_Stop(&TxTimer);
12          UTIL_TIMER_SetPeriod(&TxTimer, MAX(nextTxIn,
TxPeriodicity));
13          UTIL_TIMER_Start(&TxTimer);
14      }
15  }
16  ...
17

```


LoRa Reception Callback

The OnRxData callback is used to encode the downpayload if received otherwise the callback is not called.

The addition for the first testing procedure is shown in Listing ??.

Listing 5.13: OnRxData

```

1 static void OnRxData(LmHandlerAppData_t *appData, LmHandlerRxParams_t
   *params)
2 {
3   ...
4   MX_I2C1_Init();
5   ...
6   case LORAWAN_USER_APP_PORT:
7     if (appData->BufferSize == 1)
8     {
9       /* If the frame dimension is 1 ==> Calibration command or
10        * New Battery command is received. */
11       down_payload[0] = appData->Buffer[0];
12       if (down_payload[0] == 0x01)
13       {
14         /*New Battery Command*/
15         new_BAT_flag = 1;
16       } else if ((down_payload[0] == 0x00) &&(calibration_flag == 0))
17       {
18         /*Calibration Command*/
19         calibration_flag = 1;
20         cal_done = 0;
21
22       } else {
23         /* Update the up_payload with the error sequence
24          * to inform the use that a wrong command has
25          * been sent.*/
26         up_payload[16] = 0xFF;
27       }
28     } else if (appData->BufferSize == 3)
29     {
30       /* If the frame dimension is 3 ==> Change Max
31        * Temperature command is received. */
32       down_payload[0] = appData->Buffer[0];
33       down_payload[1] = appData->Buffer[1];
34       down_payload[2] = appData->Buffer[2];
35       if ((down_payload[0] != 0x10))
36       {
37         /* Update the up_payload with the error sequence
38          * to inform the use that a wrong command
39          * has been sent.*/
40         up_payload[16] = 0xFF;
41       }

```

```

42     }else{
43         /* If the frame has different dimension ==>
44          * Errata command is received. */
45         down_payload[0] = appData->Buffer[0];
46         up_payload[16] = 0xFF;
47     }
48     up_payload[16] = down_payload[0];
49     UTIL_TIMER_Stop(&SxTimer);
50     UTIL_TIMER_SetPeriod(&SxTimer, 1000);
51     UTIL_TIMER_Start(&SxTimer);
52     break;
53     ...
54 }

```

The code added activates a different procedure to encode the received command based on the size of the command received from user:

- if the size of the command is one than the command can be either a sequence for a new battery command or a calibration command;
- if the size is three than the command to change the maximum temperature value in the memory is received.

The code also control if the received sequence is an actual command or if it is a wrong command and it updates the sixteenth bit of the uplink frame. Then the SxTimer is stopped, its period is changed to 1s and then it is started again.

In the case of the OnRxData function, for the second procedure the only change made is the elimination of the part handling the SxTimer stop, change period and restart.

In the case of the reception callback the second procedure has the same code as the first one but only deleting the lines from 48 to 51, the rest of the code is exactly the same.

5.5.4 Additional Timer Elapse Routines

Here the callbacks of the additional timers used for both testing procedures are listed. The first three refer to the first testing procedure and the other three refer to the second testing procedure.

Sensor Simulation Timer Start Callback

This callback is shown in Listing 5.14.

Listing 5.14: OnSxTimerEvent.

```

1 static void OnSxTimerEvent(void *context)

```

```

2 {
3     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET); /* */
4     UTIL_TIMER_Start(&SxEndTimer);
5 }

```

The callback enables the pin connected to the resistor used to consumes a fixed amount of current that simulate the soil sensors consumption and to speed up the test procedure and after that it make the SxEndTimer start.

Sensor Simulation Timer End Callback

The callback is called when the seven seconds of the constant current consumption are finished in order to reset the pin connected to the resistor. The code is shown in Listing 5.15.

Listing 5.15: OnSxEndEvent.

```

1 static void OnSxEndEvent(void *context)
2 {
3     MX_I2C1_Init();
4     int exit_loops = 0;
5     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET); /* */
6     BQI2C_Gauge_Stop();
7     if (Reset != 1){
8         do{
9             exit_loops +=1;
10            HAL_Delay(500);
11            control_status = BQI2C_Control_Status();
12        } while (!(control_status & G_DONE_MASK)&& (exit_loops < 100));
13    } else
14    {
15        BQI2C_Reset();
16        BQI2C_From_Sealed_to_Unsealed();
17        BQI2C_From_Unsealed_to_FullAccess();
18        BQI2C_Eos_Init();
19        Reset = 0;
20    }
21    UTIL_TIMER_Start(&DataTimer);
22 }

```

The function first deactivates the pin and after that it send the Gauge_Stop command and it controls that the gauge goes into stop mode by controlling the G_DONE bit on the Control_Status register. If some problems occurs the gauge is reset and re-initialized. After everything the OnDataTimer is started.

Data Collection Timer Callback

The callback is used to to enter a time interval large enough to allow (according to TRM [6]) the gauge algorithm to make calculation on battery status and save data in the memory, as shown in Listing 5.16.

Listing 5.16: OnDataTimer.

```

1 static void OnDataTimer(void *context)
2 {
3     MX_I2C1_Init();
4     BQI2C_Data_Gathering_and_Payload_Creation(up_payload);
5 }

```

The only function of the callback is to gather the data and create the frame payload that must be send during next transmission.

Measurement Timer Start Callback

This callback is shown in Listing 5.17.

Listing 5.17: MeasureTimer.

```

1 static void MeasureTimer(void *context)
2 {
3     MX_I2C1_Init();
4     int exit_loops = 0;
5     if (Reset != 1)
6     {
7         BQI2C_Gauge_Start();
8         do{
9             HAL_Delay(500);
10            control_status = BQI2C_Control_Status();
11            exit_loops += 1;
12        }while (!(control_status & GAUGE_ACTIVE_MASK) && (exit_loops <
13        100));
14        if (exit_loops >= 100)
15        {
16            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);
17        }
18        else
19        {
20            HAL_Delay(1000);
21            BQI2C_Gauge_Stop();
22            do{
23                exit_loops +=1;
24                HAL_Delay(500);
25                control_status = BQI2C_Control_Status();
26            }while (!(control_status & G_DONE_MASK)&& (exit_loops < 100));
27            BQI2C_Data_Gathering_and_Payload_Creation(up_payload);

```

```

27     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);
28     }
29 } else
30 {
31     Reset = 0;
32 }
33 HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET);
34 UTIL_TIMER_Start(&BurstModeTimer);
35 }

```

The callback first control if a reset is needed and otherwise it provides as follow:

- the Gauge_Start command is issued;
- a control loop to verify if the gauge enters active mode is done;
- if the activation fail the gauge is switched off, otherwise a second of delay is applied to allow the gauge to take measurements;
- the Gauge_Stop command is issued;
- a control loop to verify if the gauge exit active mode is done;
- the data are gathered to create the payload;
- the gauge is switched off;
- the pin connected to the resistance is activated to provide the current burst;
- finally the BurstModeTimer is activated.

End Burst Timer Callback

The callback is called when one hour passes after the pin connected ti the resistance load is set to enabling the current discharge.

The code is shown in Listing 5.18.

Listing 5.18: EndBurstEvent.

```

1 static void EndBurstEvent(void *context)
2 {
3     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);
4     UTIL_TIMER_Start(&RestModeTimer);
5 }

```

It simply put the pin in reset mode and starts the rest timer in order to let the battery rest for one hour.

End Rest Timer Callback

The callback code is shown in Listing 5.19.

Listing 5.19: EndRestEvent.

```

1 static void EndRestEvent(void *context)
2 {
3     UTIL_TIMER_Start(&TxTimer);
4 }

```

The only function of the callback is to started the transmission timer.

5.6 Decoder for TTN

Live data from The Things Network console can show the received frame payload but, if you want to know the meaning of it is needed to write a decoding code. An easy way to do it is to write it in Javascript. In Listing 5.20 is shown the employed code based on what is programmed in the firmware code of the MCU.

Listing 5.20: Decoder fot TTN.

```

1 function Decoder(bytes, port) {
2     var bitMapping = {
3         0x01: "Battery Low",
4         0x02: "Temperature High",
5         0x04: "Temperature Low",
6         0x08: "EOS"
7     };
8     var decoded = {};
9     if (port == 2) //runtime port
10    {
11        var Bq_condition = bytes[0];
12        var risultato = "";
13        if (Bq_condition == 0) {
14            risultato = "OK ";
15        } else {
16            for (var bit in bitMapping) {
17                if (Bq_condition & parseInt(bit)) {
18                    risultato += bitMapping[bit] + ", ";
19                }
20            }
21        }
22        if (risultato.length > 0) {
23            risultato = risultato.slice(0, -2);
24        }
25        decoded._1Bq_condition = risultato;
26        var Voltage = (bytes[1]<< 8) | (bytes[2]);

```

```

27 |     decoded._2Voltage = Voltage/1000;
28 |     var Temperature = (bytes[3]<< 8) | (bytes[4]);
29 |     decoded._3Temperature = (Temperature/10) - 273.15;
30 |     var Current = (bytes[5]<< 8) | (bytes[6]);
31 |     decoded._4Current = complementToDecimal(Current, 16);
32 |     var Scaled_R = (bytes[7]<< 8) | (bytes[8]);
33 |     decoded._5Scaled_R = Scaled_R;
34 |     var Impedance_Z = (bytes[9]<< 8) | (bytes[10]);
35 |     decoded._6Impedance_Z = Impedance_Z;
36 |     var SOH = bytes[11];
37 |     decoded._8SOH = SOH;
38 |     var Status = bytes[13];
39 |     decoded.Status = Status.toString(2);
40 |     var Alert = bytes[12];
41 |     decoded.Alert = Alert.toString(2);
42 |     var Cnt = (bytes[14]<< 8) | (bytes[15]);
43 |     decoded.Cnt = Cnt.toString(2);
44 |     var last_command = bytes[16];
45 |     decoded.last_command = last_command;
46 |     return decoded;
47 | }
48 | }

```

After an if-condition to select the right port (fixed port in this application), the decoder evaluates all data read from the gauge and sent to the server.

The decoder takes the payload from the LoRa message and divides it in the information that represents:

- **Byte[0]** represents the gauge conditions converted in string message according on the value;
- **Byte[1] e Byte[2]** represent the voltage measured by the gauge in mV and reconverted into V for user view;
- **Byte[3] e Byte[4]** represents the external temperature measured by the gauge in 0.1K and reconverted in Celsius;
- **Byte[5] e Byte[6]** represent the last current sample measured by the gauge in mA;
- **Byte[7] e Byte[8]** represents the internal resistance of the battery calculated by the gauge in $m\Omega$;
- **Byte[9] e Byte[10]** represents the internal impedance of the battery calculated by the gauge in $m\Omega$;
- **Byte[11]** represents SOH percentage calculated by the gauge;

- **Byte[12]** represents the Battery_Status register presented as sequence of bit;
- **Byte[13]** represents the Battery_Alert register presented as sequence of bit;
- **Byte[14] e Byte[15]** represents the Control_Status register presented as sequence of bit;
- **Byte[16]** represents the last command received by the user.

Chapter 6

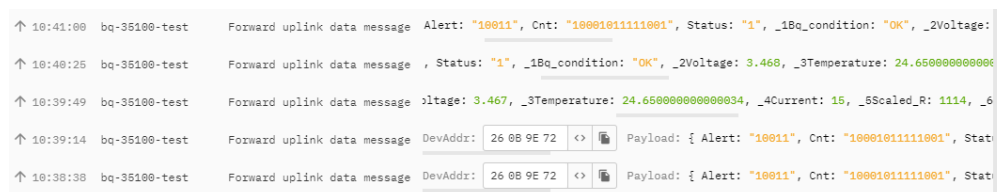
Experimental Results

This chapter will presents the results of the two tests and some consideration regard how much the test can be reliable for the battery under test.

For the purpose of the project the reliability of the test is the ability to detect properly the change in the internal resistance of the battery because it is the only way to predict when a LiSOCl² battery comes close to the end-of-life condition.

6.1 Data Managing

During the whole tests the data are constantly monitored with TTN Console Network. An example of the data coding is presented below in figure 6.1



```
↑ 10:41:00 bq-35100-test Forward uplink data message Alert: "10011", Cnt: "10001011111001", Status: "1", _1Bq_condition: "OK", _2Voltage:
↑ 10:40:25 bq-35100-test Forward uplink data message , Status: "1", _1Bq_condition: "OK", _2Voltage: 3.468, _3Temperature: 24.650000000000
↑ 10:39:49 bq-35100-test Forward uplink data message Voltage: 3.467, _3Temperature: 24.65000000000034, _4Current: 15, _5Scaled_R: 1114, _6
↑ 10:39:14 bq-35100-test Forward uplink data message DevAddr: 26 08 9E 72 <> Payload: { Alert: "10011", Cnt: "10001011111001", Stat
↑ 10:38:38 bq-35100-test Forward uplink data message DevAddr: 26 08 9E 72 <> Payload: { Alert: "10011", Cnt: "10001011111001", Stat
```

Figure 6.1: TTN decoded samples from TTN console network.

The data for both testing procedures uses the same decoder, so they were presented in the same way.

6.2 First Test

The test ended as expected took 13 days to have a complete discharge of the battery and obviously for the the MCU to stop sending messages.

During almost the whole test the measurements where taken every 35 seconds

ending up to have approximately 30000 samples for all test.

For this test data for voltage measurement shows a behaviour as expected, as the behaviour of the battery cell voltage is shown in figure 6.2. The profile of the

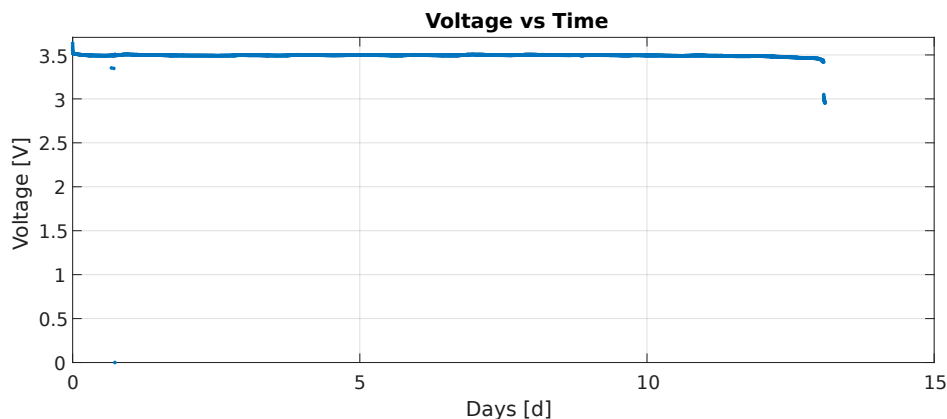


Figure 6.2: Voltage vs Time (First Test).

battery voltage is very very flat for almost the entire test and it starts decreasing only six hours before the battery died completely, as expected for LS14500 battery. Indeed the measured voltage from the test shows almost the same behaviour as the one described in the LS14500 datasheet [3] and shown in figure 2.2.

Despite of the voltage being measured properly, other parameters measured present an unexpected behaviour.

For example the State of Health (SOH, shown in figure 6.3) determined by the BQ35100 exhibits a consistent 96% level for the majority of the test duration, except for the fact that it starts to decline steadily during the final 6 hours of the test, eventually reaching 0% just before the battery discharges completely and the MCU sends the last message.

The other important value that presents an unusual behavior is the internal resistance calculated by the gauge algorithm.

The value remains quite always fixed value after very few samples instead of growing drastically, as said in EOS configuration manual [8], when the battery is at 50% of discharge, as in figure 3.5. This behaviour is shown in figure 6.4. In order to explain this strange behaviour it must be consider two things that are explained in BQ35100 datasheet [5]:

1. after receiving the gauge stop and asserted the G_DONE bit the gauge should be turned off;
2. the gauging algorithm made calculation on resistance and SOH starting from the battery voltage measured each cycle when the gauge received a

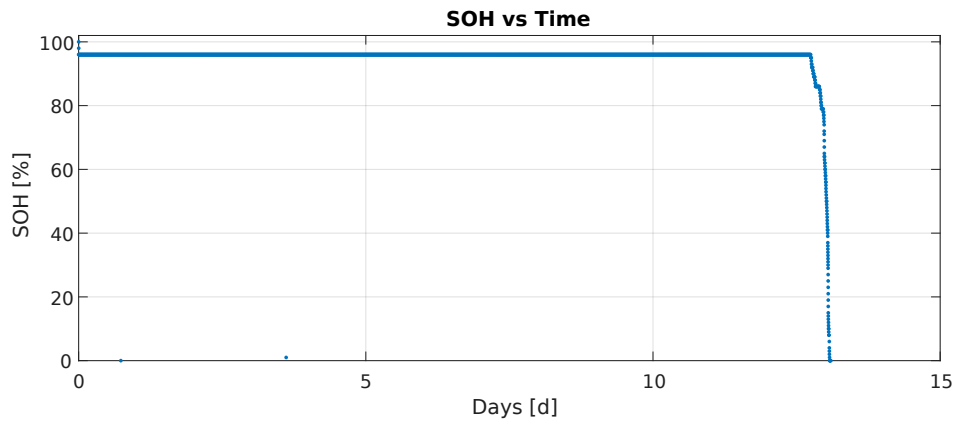


Figure 6.3: State Of Health Behaviour vs Time (First Test).

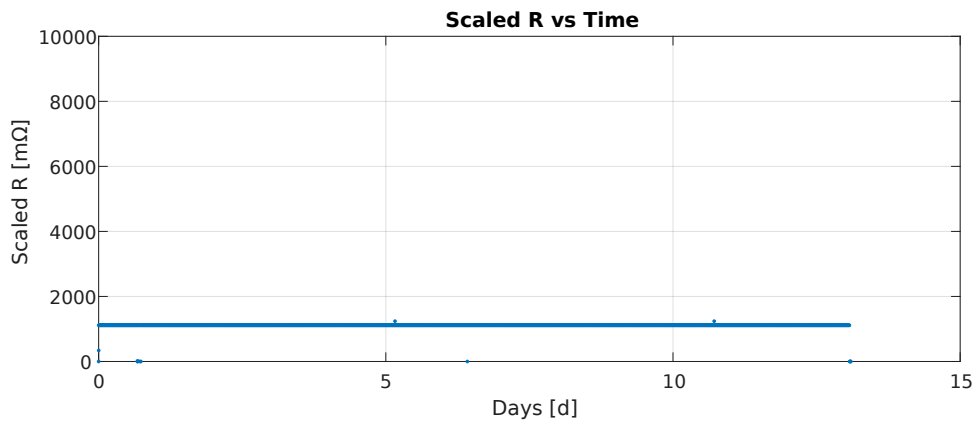


Figure 6.4: Internal Resistance Behaviour vs Time (First Test).

Gauge_Start command, so in order to take accurate calculation of the resistance the battery needs a proper time to rest in order to restore its voltage as close as possible to the first one measured during first cycle.

That said there are some possible problems related to the first testing procedure:

- the battery had never get enough rest time so the voltage is always a little lower the first voltage measure each cycle is lower so the average resistance calculated remains the same each time, so its value is constant and the same concept can be extended also to the SOH behaviour;
- the manuals regarding the BQ35100 does not show any example of the case where the gauge is used without being powered off, even if it is said that it can be used either this way; so there is a possibility that some adjustments are needed that are not specified in the manuals;

- another possibilities is that the time the gauge used to do the calculations given by the manuals (15 sec) is too short considering that the interval of time to gather measurements is at least 14 seconds, so maybe this value should be changed in data flash.

6.3 Second Test

For the second test procedure the duration of the test does not last exactly four days but some hour less.

Unlike the previous test in this test all data read from the gauge presents a behaviour quite close to the expected and desirable one.

As proof of the correctness of the results obtained the State-Of-Health of the battery decreases linearly during time as the test progressed, as shown in figure 6.5.

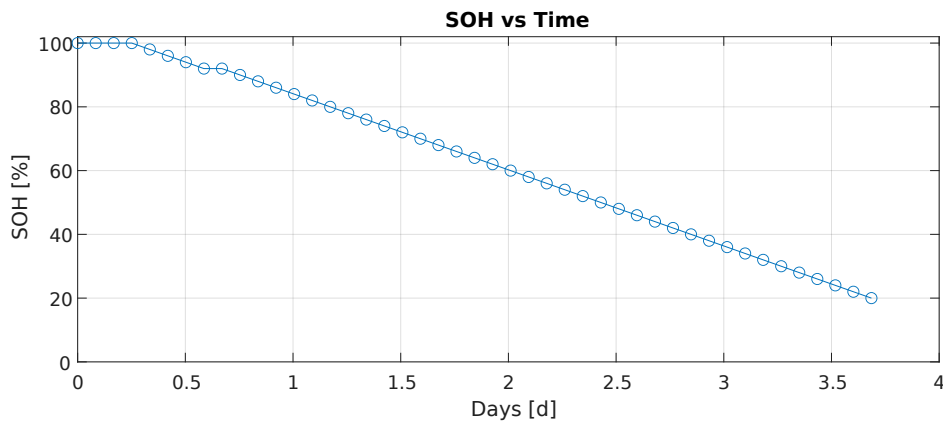


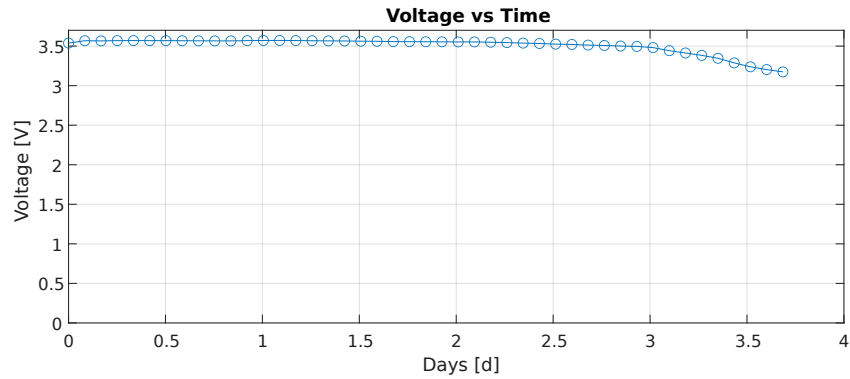
Figure 6.5: State Of Health Behavior During Time (Second Test).

The figure shows that the SOH decreased each time it is sampled from the battery gauge as it would be expected in a normal behaviour, but also a very important fact is that the SOH decreases linearly.

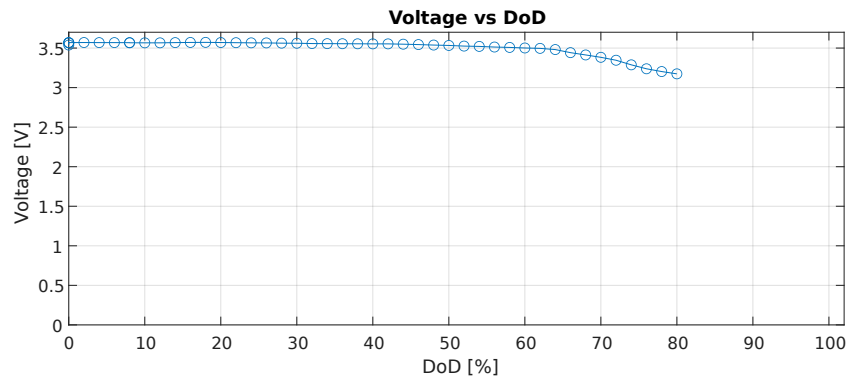
The first samples show the same value because firstly the BQ35100 considers that LiSOCl_2 presents a passivation layer that is removed during the first three burst consumption and in order to take this phenomenon into account the gauge automatically considers the SOH the same for the first samples.

Taking into account that in this test the measurement of the SOH is reliable for the performance analysis a more accurate analysis on the parameters can be done. In particular in this case is possible to correlate also the voltage and the internal resistance not only with time, but also with respect to the Depth-of-Discharge (DoD).

The battery voltage behaviour monitored during the test is shown in figure 6.6.



(a) Voltage Behaviour vs Time (Second Test).



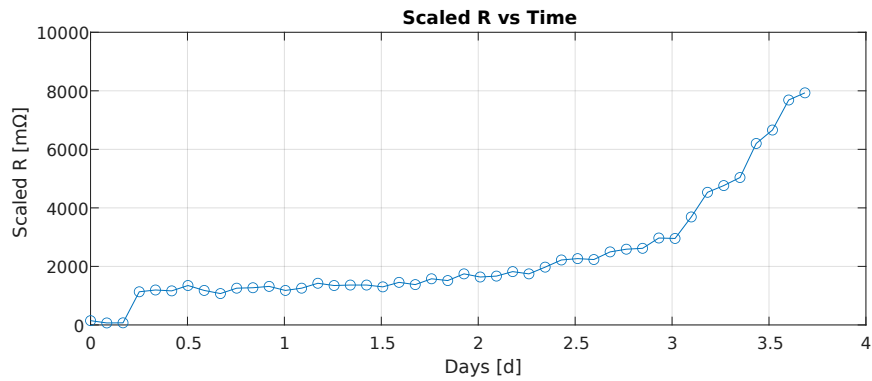
(b) Voltage Behaviour vs Depth Of Discharge (Second Test).

Figure 6.6: Voltage Behaviour respect to Time and Depth-of-Discharge (Second Test).

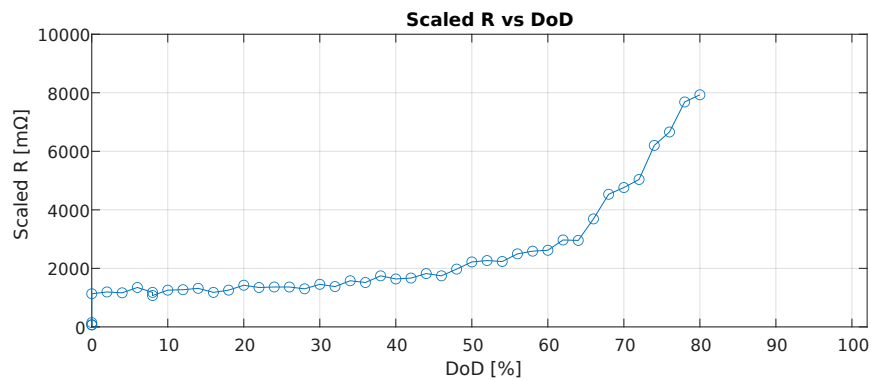
As shown in figure 6.6a the battery voltage is the one expected from an LS14500, but in this case the battery discharges during the burst consumption before the last message with the last measurement taken from the gauge so the last data are missing. The same can be said for the battery voltage behaviour respect to DoD in figure 6.6b, in this case the voltage monitoring ends when the SOH reaches 80%, the trend of the voltage is flat between 0%-60% of DoD and decreases very slowly after that point.

During this test a quite remarkable result can be seen by looking at the behaviour of the internal resistance measured by the BQ35100.

To analyze correctly the internal resistance behavior it must be considered when related to the time but more important when considered respect to the DoD. The figures showing this relation are presented in figure 6.7.



(a) Behaviour of Internal Resistance vs Time (Second Test).



(b) Behaviour of Internal Resistance vs DoD (Second Test).

Figure 6.7: Internal Resistance Behaviour respect to Time and Depth-of-Discharge (Second Test).

The plots presented above in both figures 6.7a and 6.7b shows the exact same behaviour as the one shown in figure 3.5. So in this case the internal resistance measured by the gauge algorithm shows a correct behaviour.

In particular:

- firstly it shows a flat behaviour with the resistance value almost stuck below $2000m\Omega$;
- after the gauge reaches 50% of DoD then the resistance starts to increase until reaching the final value before the battery discharges completely.

Chapter 7

Hardware Development

For completion reasons a PCB project is developed with the battery gauge involved. The project is involved in a project different from WAPPFRUIT, but it is also related to a smart-agriculture system powered by LS14500 battery that provides measurements of environment humidity and temperature.

7.1 Components Required

The first design in a PCB project is the definition of the materials needed and the analysis of the electrical and dimensional requirements of each one in a PCB design.

Lorato Module

This is a system based on STM32-WL55 MCU developed in a parallel project by the eLiONS research group, the 3D view of the component is shown in figure 7.1.

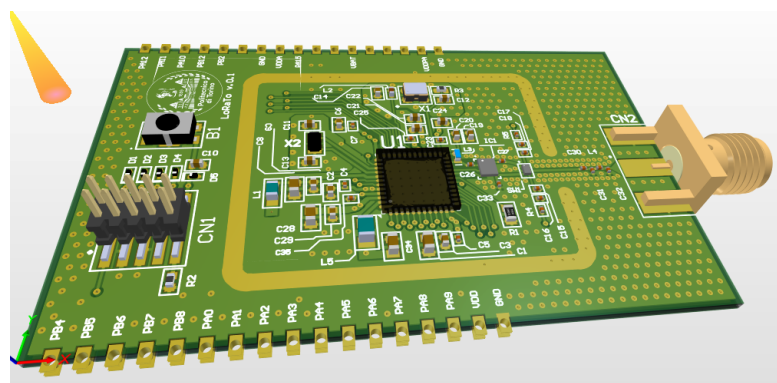


Figure 7.1: 3D Model of LoRaTO Module.

The Lorato Module is equipped also with an RF module that is used to implement and to handle the LoRaWAN communication protocol.

Battery Management: BQ35100

The battery gauge used for calculating the near battery end-of-life condition. The chip is shown figure 7.2.



Figure 7.2: BQ35100-PWR Chip.

Temperature and Humidity Sensor

Regarding the choice of the sensor, the best choice is to use the HDC302x family from Texas Instruments, and the sensor chosen for the task is the HDC3022 sensor by Texas Instruments shown in figure 7.3.

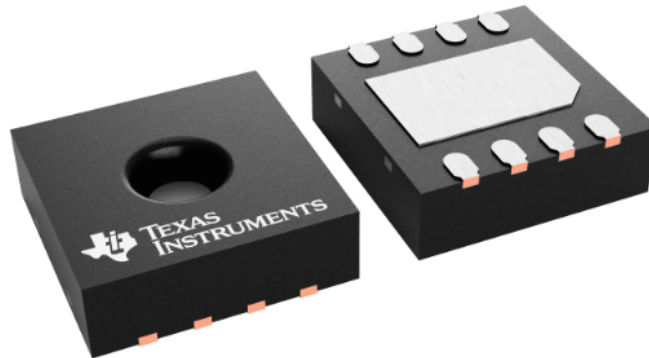


Figure 7.3: HDC3022 Temperature and Humidity Sensor Chip (Source [16]).

The HDC3022 is an integrated humidity and temperature sensor that provides high accuracy measurements with very low power consumption and programmable interrupt thresholds to provide alerts and system wake-ups without requiring a

MCU to be continuously monitoring the system and also it has a very low current consumption and a condensation protection feature with integrated heater.

The self-heating ability is what poses the most critical problem because it requires a very large amount of current when activated and it can last for more than 10 seconds, so it can be an issue for the compatibility with LS14500.

The data from the sensor can be available through I²C interface for the MCU to read.

Due to the higher thermal conductivity of copper, running solid ground planes between other components on the board and the sensor will cause undesired heat transfer, so it is best to avoid copper planes near the sensor that are connected to the copper planes of other components on the board.

LDO Voltage Regulator

One very important aspect to take into account for minimizing the power consumption of smart agriculture systems is the low standby consumption. When the device does not have to collect data from sensors, the MCU must stay in stop mode.

It is not recommended to power the MCU using directly the battery voltage because it can expose the core of the system to several possible problems (dead battery, high loads, spike of voltage) depending on the nature of an electrochemical battery. As that said it is recommended to power the MCU with a regulator in order to guarantee a stable voltage without possible stranger behaviours. The most common value as supply voltage for this kind of systems is 3.3 V, considering also the MCU and sensors.

The most reasonable choice for this scope is to use a LDO (Linear Dropout) STLQ020 Voltage Regulator of the STMicroelectronics [17]. The scheme of the component is shown in figure 7.4. This regulator has a maximum dropout voltage of 160 mV (with maximum load of 200 mA) but also a very low quiescent current (300 nA with no load), that are very important for keeping stand-by consumption as low as possible. The regulator is also available in SOT323-5L package that is very convenient for this latter model.

Load Switches

In order to avoiding unwanted current going through resistors for the calibration of the battery gauge, it is very important to isolate the resistance.

A very good choice is to use a load switch in a pass transistor configuration and for this task the SIP32431 p-channel Pass Transistor from Vishay [18] directly controlled by the MCU. The scheme of the component is shown in figure 7.5.

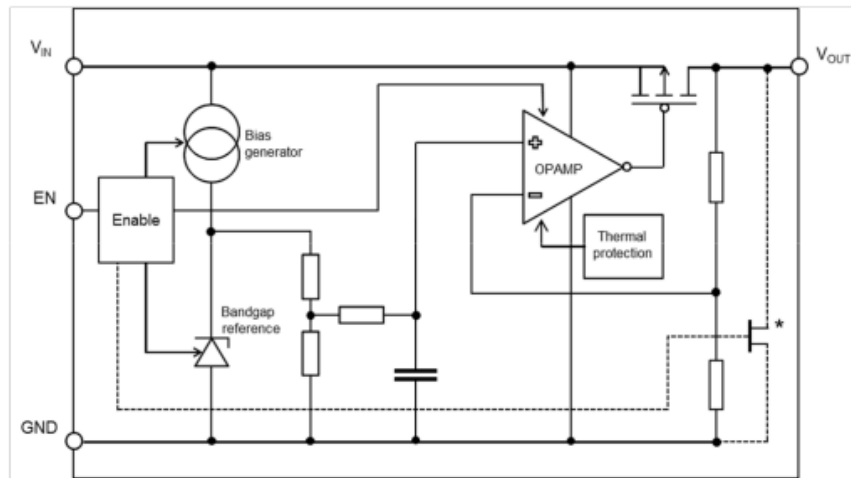


Figure 7.4: STQL020C33R Voltage Regulator (Source [17]).

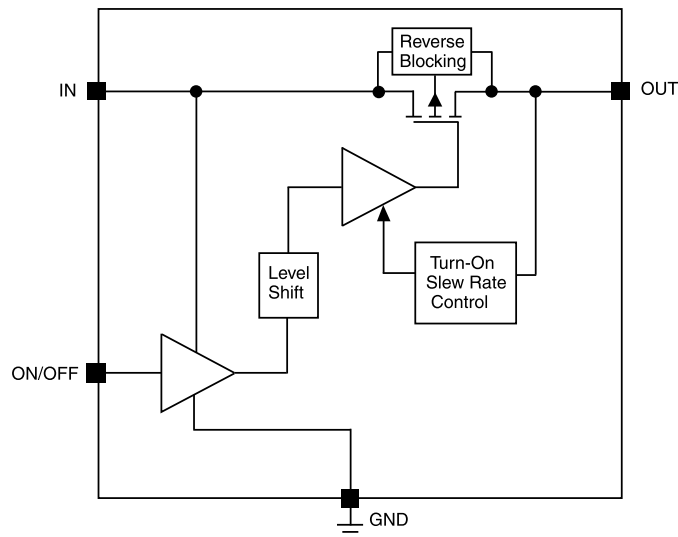


Figure 7.5: SIP32431 P-channel Pass Transistor (Source [18]).

In this way it is possible to keep the stand-by consumption to 10 pA, when the system does not calibrate the gauge.

Passive Components

The passive components are important for the system because they provide noise filtering, constant current consumption and so on.



Figure 7.6: Example of SMD Capacitors.

In the choice of the passive components (as resistances and capacitors) a very important feature to take into account for a PCB design is the dimension of the component because it directly affect the overall size of the board and consequently also the cost of the system.

Battery Holder

Last but not less important is the choice of a battery holder suitable for the LS14500.

The best choice would be a battery holder with a lower part opened in order to place the NTC thermistor in the middle to keep track constantly of the battery temperature, but that it is also product with SMD connections.

For this purpose the battery holder chosen is the 1024TR AA-size from Digikey, which is shown in figure 7.7.



Figure 7.7: 1024TR AA-size Battery Holder.

Even if the battery holder has no opening for placing the NTC thermistor, an hole will be made before assembling it.

Over-current Protection

- PTC Reset Fuse: The 1210L Series PTC provides surface mount overcurrent protection for applications where space is at a premium and resettable protection is desired.



Figure 7.8: 1210L Series PTC RESET (Source [19]).

- Polarity Protection Diode: for reverse polarity protection in order to protect the device from a miswired input, such as a reversed battery, the best option is to use the LM66100 from Texas instrument, shown in figure 7.9.

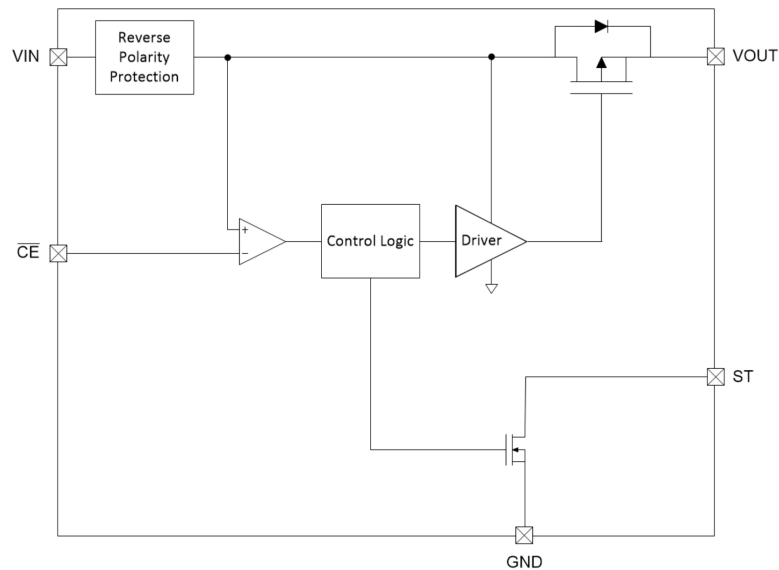


Figure 7.9: LM66100, Low IQ Ideal Diode With Input Polarity Protection.

As stated in [20] the LM66100 is a Single-Input, Single-Output integrated ideal diode that contains a P-channel MOSFET which can operate over an input voltage range of 1.5 V to 5.5 V and can support a maximum continuous

current of 1.5 A. The chip enable works by comparing the CE pin voltage to the input voltage. When the CE pin voltage is higher than V_{IN} , the device is disabled and the MOSFET is off. When the CE pin voltage is lower, the MOSFET is on.

7.2 Schematic

The next step is the realization of a schematic that connects all the components together and describes all the electrical connections. The schematic sheet of the PCB project produced is shown in figure 7.10.

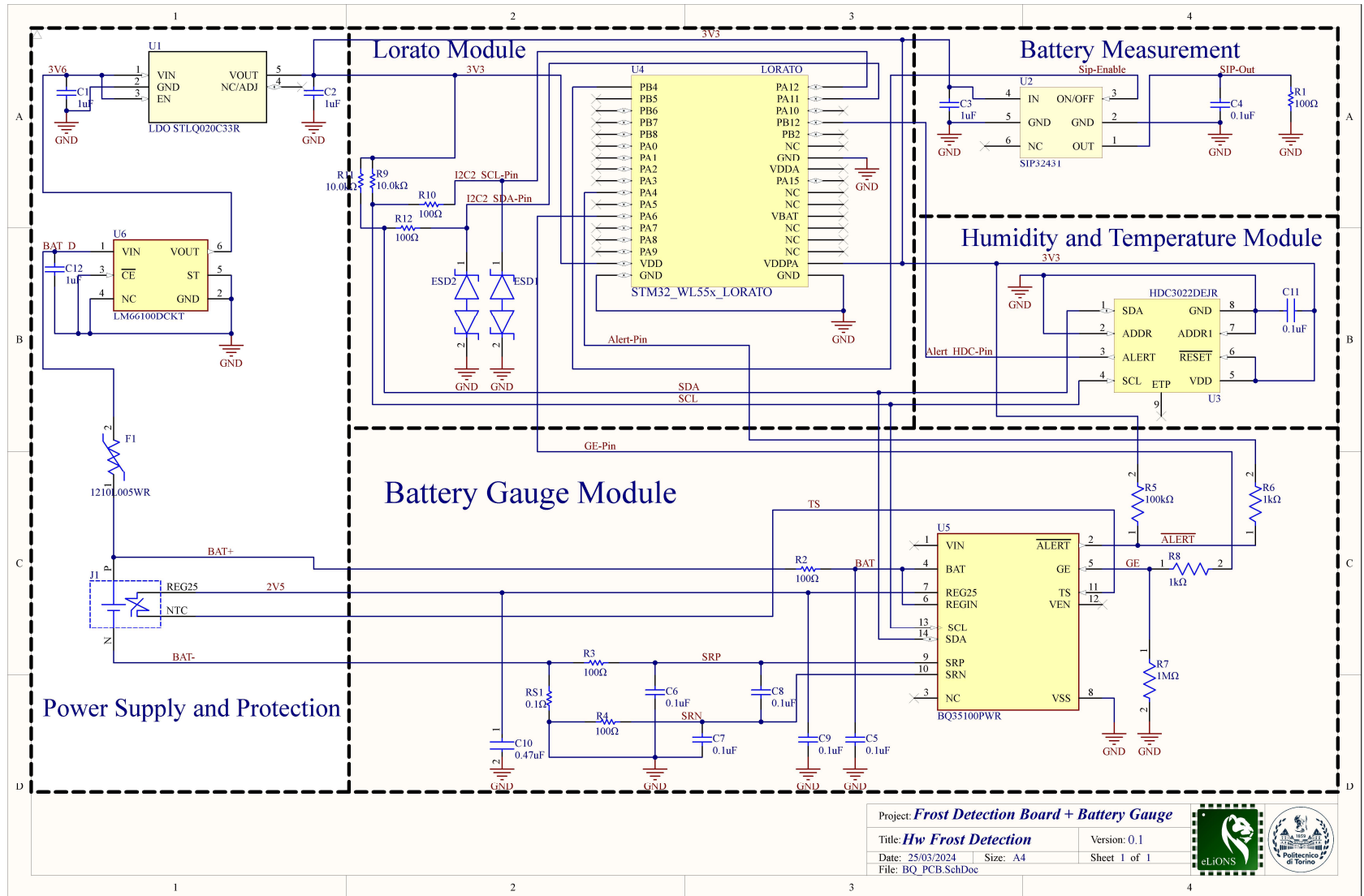


Figure 7.10: Schematic Sheet.

7.3 PCB Layout

When all logical connections are issued in the schematic, the last step is the design of the PCB Layout.

This step is important because it has to satisfy geometry constrains, physical constrains and reliability constrains.

For this project the most important constrains is related to the fact that it is necessary to leave place where the Lorato module will be assembled (in the blue rectangle) without any other component.

In this project both top layer and bottom layer In figure 7.11 the different planes on the board are highlighted.

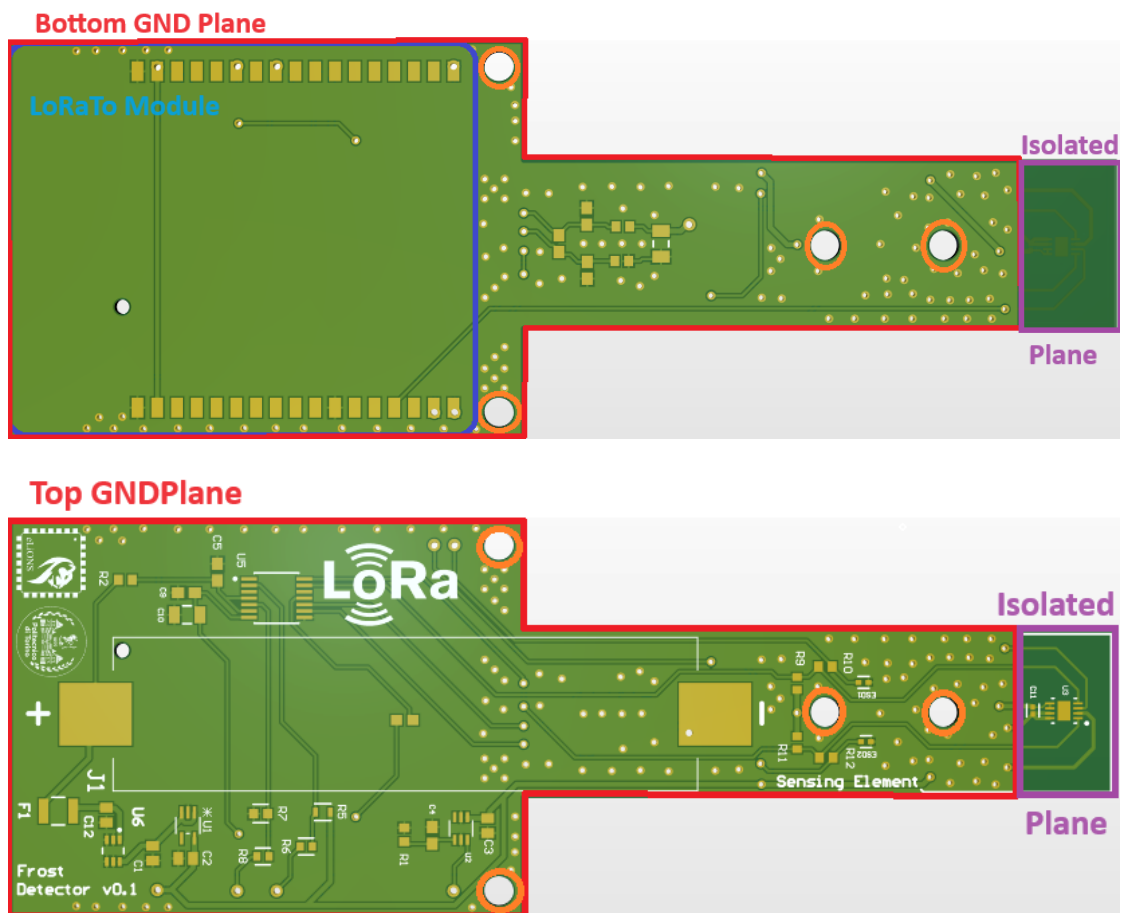


Figure 7.11: Top and Bottom View with Highlighting on Different Zones.

As shown there is a specific isolated plane where the temperature and humidity sensor (highlighted in purple) will be placed in order to avoid heat from current

flowing through the ground plane.

Multiple holes (highlighted in orange) are placed in order to insert the external support for the case.

7.4 Bills Of Materials

Designator	Comment	Description	Quantity
C1, C2, C3, C12	GRM21BR71H105KA12L	1uF Capacitor	3
C4, C5, C6, C7, C8, C9	06033D104KAT2A	0.1uF Ceramic Capacitor, Multilayer, Ceramic, X5R, 15% TC, 0.1uF, Surface Mount, 0603	7
C10	106033D104KAT2A	0.1uF Ceramic Capacitor, Multilayer, Ceramic, X5R, 15% TC, 0.1uF, Surface Mount, 0201	1
C11	12065C474JAT2A	0.47uF General Purpose Ceramic Capacitor, 1206, 470nF, X7R, 15%, 50V	1
ESD1, ESD2	TPD1E10B06DPYR	TVS DIODE 5.5V 14V 2X1SON	2
J1_Battery_Holder	Battery Holder	1024TR: 1 Cell AA-size Battery Holder	1
J1_NTC	NT06104F3435B1F	Thermistor NTC 10k 0603	1
R1, R2, R3, R4, R10, R12	CRCW0603100RFKEA	Resistor 100Ω, 100 mW, -55 to 155 degC, 0603 (1608 Metric)	6
R5	CRCW0603100KFKEAC	Resistor 100kΩ SMD, 1/10W, 0603	1
R6, R8	CRCW06031K00JNEA	Resistor 1kΩ Thick Film, 0.1W, 0603	2
R7	CRCW06031M00JNEA	Resistor 1MΩ Thick Film, 0.1W, 0603	1
R9, R11	CRCW060310K0FKEA	Resistor 10.0kΩ Chip Resistor, 100 mW, -55 to 155 degC, 0603 (1608 Metric)	2
RS1	WSL1206R1000FEA	0.1 Ohm sense resistor	1
U1	LDO STLQ020C33R	Voltage Regulator -Output 200mA SOT-323-5, IC REG LIN 3.3V 200MA SOT323-5	1
U2	SiP32432	"p-channel MOS Pass Transistor 10 pA, Ultra Low Leakage and Quiescent Current, Load Switch with Reverse Blocking"	1
U3	HDC3022	Serial Switch/Digital Temperature and Humidity Sensor, 14 Bit(s), 0.40Cel, Square, Surface Mount	1
U4	LORATO	LORATO Module	1
U5	BQ35100PWR	IC BATTERY MONITOR 14TSSOP	1
U6	LM66100DCKT	Ideal diode with Integrated FET 6-SC70 -40 to 105 1.5-V to 5.5-V, 1.5-A, 0.5-uA IQ	1
F1	FEMTOSMDC005F-2	Resettable fuse PTC Polymeric 15V 50 mA 1h Surface mount 0603 (1608 metric), Concave, time to trip 0.1 sec, power dissipation 0.5W	1

Table 7.1: Bill Of Materials.

7.5 3D Models

The 3D-model of the complete system of the frost detection board with the Lorato module is presented in figure 7.12.

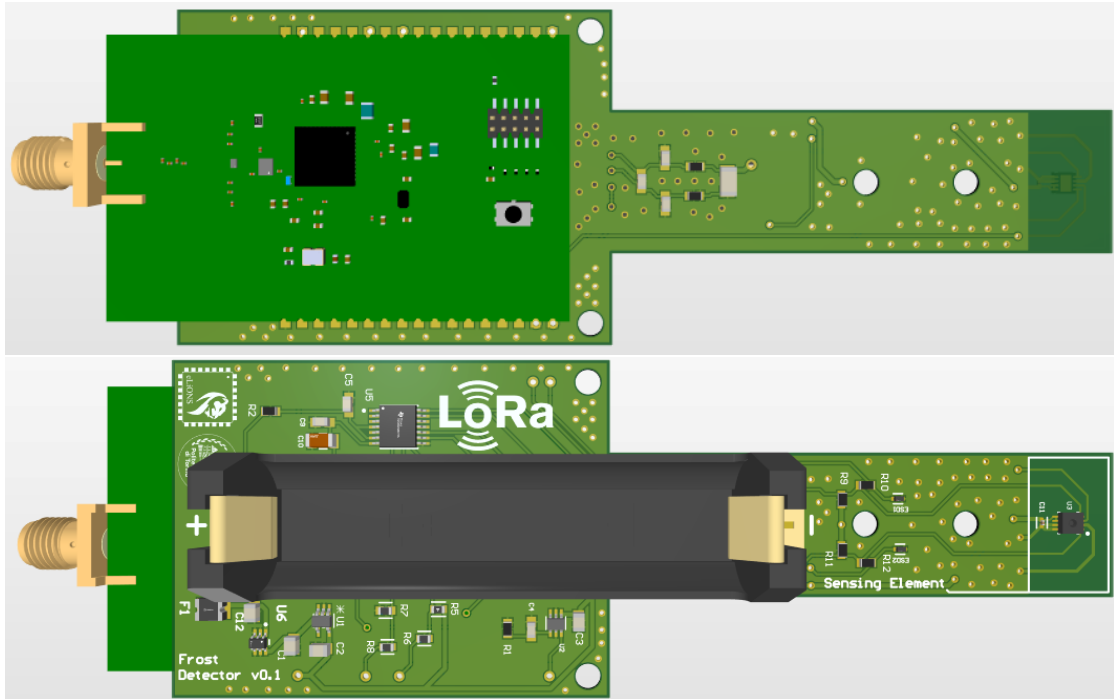


Figure 7.12: Top 3D view and Bottom 3D view of the PCB.

7.6 Manufacturing

Manufactured printed circuit board assembled with all components and LoRaTO module is shown in 7.13.



Figure 7.13: Top view and Bottom view of the Frost Detection System assembled.

Chapter 8

Conclusion and Future Perspective

The main perspective of this project to develop a system (firmware and hardware) for smart-agriculture systems in which the management of the proper time to change the battery when it comes near to the end-of-life condition and to integrated it in a system that has different task to achieve.

In particular the experimentation done with the battery gauge allows to establish that the best solution for battery gauging well being is to use a separate learning pulse to take correct measurements and evaluate properly the battery status, because the measurements of the BQ35100 taken during the stress test reflect almost perfectly the same results as the ones reported in the manuals.

There are also some aspects that can further exploited in order to make better evaluations on the system developed for better fit in the smart-agriculture systems. For example a proper evaluation of the firmware should be made with a real case scenario discharge test, this because the amount of time the battery would rest before taking any measure is far greater than the one used in test procedure so the computation of battery replacement can be more precised as the estimation of state of charge.

Regarding this aspect, the software to drive the developed board will be perfected and the board will be tested in an on-field test during the 2024 in order to complete the system.

Bibliography

- [1] eLiONS, Politecnico di Torino. *WAPPFRUIT – Intelligent Technologies Applied to Water Management in Fruit Cultivation*. URL. 2014-2020 (cit. on p. 1).
- [2] LoRa Alliance. *LoRa documentation*. URL: <https://lora.readthedocs.io/en/latest/#> (cit. on pp. 2, 4, 5).
- [3] *Primary lithium battery LS 14500*. URL. Saft. 2009 (cit. on pp. 6–8, 60).
- [4] *Description of STM32WL HAL and low-layer drivers*. STMicroelectronics. URL: https://www.st.com/content/ccc/resource/technical/document/user_manual/group1/6f/be/85/55/8c/26/4c/22/DM00660673/files/DM00660673.pdf/jcr:content/translations/en.DM00660673.pdf (cit. on pp. 8, 9).
- [5] *BQ35100 Lithium Primary Battery Fuel Gauge and End-Of-Service Monitor datasheet (Rev.E)*. Texas Instrument. 2019. URL: https://www.ti.com/lit/ds/symlink/bq35100.pdf?ts=1695628913180&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FBQ35100 (cit. on pp. 10–14, 16, 17, 60).
- [6] *bq35100 Technical Reference Manual (Rev.C)*. Texas Instrument, 2018. URL: https://www.ti.com/lit/ug/sluubh1c/sluubh1c.pdf?ts=1695586329243&ref_url=https%253A%252F%252Fchina.ti.com%252Fsupport%252Fmachine-translation%252Fmt-power-management%252Ff%252Fmt-power-management-forum%252F310729%252Fbq35100-bq35100-ti%252F1040390 (cit. on pp. 12, 13, 17, 20, 25, 26, 29, 33, 45, 54).
- [7] *Alternative Gauging Techniques for Flow Meters and the Benefits of the bq35100*. Texas Instrument. 2018. URL: https://www.ti.com/lit/an/slua904/slua904.pdf?ts=1695676027012&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FBQ35100%253FkeyMatch%253DBQ35100%2526tisearch%253Dsearch-everything%2526usecase%253DGPN (cit. on p. 15).

- [8] *How to Configure the BQ35100 for EOS Mode*. Texas Instrument. 2022. URL: https://www.ti.com/lit/an/sluaal7/sluaal7.pdf?ts=1695675921331&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FBQ35100%253FkeyMatch%253DBQ35100%2526tisearch%253Dsearch-everything%2526usecase%253DGPN (cit. on pp. 15, 39, 60).
- [9] *Using I2C Communications With the bq34110 bq35100 and bq34z100-G1 Series of Gas*. Texas Instrument. 2016. URL: https://www.ti.com/lit/an/slua790/slua790.pdf?ts=1695665499672&ref_url=https%253A%252F%252Fwww.google.com%252F (cit. on pp. 20, 23).
- [10] *Gauge Communication*. Texas Instrument. 2017. URL: https://www.ti.com/lit/an/slua801/slua801.pdf?ts=1695617690314&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252Fko-kr%252FBQ40Z80 (cit. on p. 30).
- [11] M. Barezzi et al. «Long-Range Low-Power Soil Water Content Monitoring System for Precision Agriculture». In: *2022 IEEE 13th Latin America Symposium on Circuits and System (LASCAS)*. Puerto Varas, Chile, 2022, pp. 1–4. DOI: 10.1109/LASCAS53948.2022.9789070 (cit. on p. 36).
- [12] M. Barezzi et al. «Long-Range Low-Power Electronic System for Drip Irrigation in Precision Agriculture». In: *2023 IEEE Conference on AgriFood Electronics (CAFE)*. Torino, Italy, 2023, pp. 167–171. DOI: 10.1109/CAFE58535.2023.10291511 (cit. on pp. 36, 37).
- [13] *Using the bq35100 with Li-Primary Based Applications*. Texas Instrument. 2018. URL: https://www.ti.com/lit/an/slua904/slua904.pdf?ts=1695676027012&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FBQ35100%253FkeyMatch%253DBQ35100%2526tisearch%253Dsearch-everything%2526usecase%253DGPN (cit. on p. 39).
- [14] *bq35100EVM-795 Evaluation Module User's Guide*. Texas Instrument. 2016. URL: https://www.ti.com/lit/ug/sluubh7/sluubh7.pdf?ts=1695677832988&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FBQ35100%253FkeyMatch%253DBQ35100%2526tisearch%253Dsearch-everything%2526usecase%253DGPN (cit. on p. 42).
- [15] STMicroelectronics. *STM32Cube MCU Package examples for STM32WL Series*. URL. Last update 2023 (cit. on p. 42).
- [16] Texas Instruments. *HDC3020 Datasheet*. December 2022. URL: https://www.ti.com/lit/ds/symlink/hdc3020.pdf?ts=1706169374654&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FHDC3020%253FkeyMatch%253DHDC3020%2526tisearch%253Dsearch-everything%2526usecase%253DGPN-ALT (cit. on p. 66).

BIBLIOGRAPHY

- [17] STMicroelectronics. *STLQ020 Datasheet*. October 2021. URL: <https://www.st.com/resource/en/datasheet/stlq020.pdf> (cit. on pp. 67, 68).
- [18] Vishay. *SIP32431 Datasheet*. July 2020. URL: <https://www.vishay.com/docs/66597/sip32431.pdf> (cit. on pp. 67, 68).
- [19] Littelfuse. *1210L Series Datasheet*. 2024. URL: <https://www.littelfuse.com/media?resourcetype=datasheets&itemid=b3a2be92-83a1-491d-9c6d-dc64451de047&filename=1210l-datasheet-update> (cit. on p. 70).
- [20] Texas Instruments. *LM66100 Datasheet*. June 2019. URL: https://www.ti.com/lit/ds/symlink/lm66100.pdf?ts=1705312697427&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FLM66100%253Futm_source%253Dgoogle%2526utm_medium%253Dcpc%2526utm_campaign%253Dapp-null-null-gpn_en-cpc-pf-google-eu%2526utm_content%253Dlm66100%2526ds_k%253DLM66100%2526dcm%253Dyes%2526gad_source%253D1%2526clid%253DCjwKCAiAzJOtBhALEiwAtwj8ttfdMvD_7lrJJ90G6cEtaXuMbs4P7eLPJRgBvhvJuhHHekWn-U0ixoC0cEQAvD_BwE%2526gclidsrc%253Daw.ds (cit. on p. 70).