CELL-INTEGRATED SENSING FUNCTIONALITIES FOR SMART BATTERY SYSTEMS WITH IMPROVED PERFORMANCE AND SAFETY

GA 957273

D4.2 - BMS-MASTER SOFTWARE ENVIRONMENT

LC-BAT-13-2020 - Sensing functionalities for smart battery cell chemistries

Deliverable No.	D4.2	
Related WP	4	
Deliverable Title	BMS-master software environment	
Deliverable Date	29-11-2022	
Deliverable Type	OTHER	
Dissemination level	Public (PU)	
Written By	Juan José Jurado Laguna (IKE) Igor Pérez de Arenza (IKE)	18-11-2022 18-11-2022
Checked by	Joris De Hoog (FM)	29-11-2022
Reviewed by	Martin Wenger (FHG) Joris De Hoog (FM)	23-11-2022 23-11-2022
Approved by	Iñigo Gandiaga (IKE)	29-11-2022
Status	Final	29-11-2022

Summary

The main objective of this deliverable is to present the software environment to control a novel BMS-Master design based on a rapid prototyping platform compatible with the advanced BMS slaves presented in the D4.1. The BMS-master will allow a reliable and optimal use of the capabilities based on improved accuracy state algorithms (to be developed in Task 4.4).

The developed BMS-master evaluates the measurement information coming from the BMS-slave(s) which can read-out the cell-integrated SENSIBAT Level-1 sensor, as well as the cell voltages and carries out the operations to control a module consisting of 6 series-connected cells using the previously designed Power Measurement and Disconnection Unit.

The developed BMS-master hardware board is based on a rapid prototyping platform composed by a Raspberry PI 4 model B and an Arduino Due. For example, using the developed BMS Master it will be possible to quickly make multiple versions of the SENSIBAT SoX algorithms and begin testing or validating quicker. In summary, the rapid prototyping of a BMS allows software designers to give an accurate concept of how the finished product will work out before putting too much time and money into the prototype.

This deliverable does not include any deviation from the objectives and timings planned in the Grant Agreement of the project.

Table of Contents

Table of Figures

Abbreviations

Abbreviation	
APP	*Application*
ARD	*Arduino Due*
BMS	*Battery Management System*
BMS-M	*BMS-Master*
CAN	*Controller Area Network*
HMI	*Human Machine Interface*
LabVIEW	*Laboratory Virtual Instrument Engineering Workbench*
PCB	*Printed Circuit Board*
PMDU	*Power Measurement and Disconnection Unit*
RPI	*Raspberry PI 4 model*
SLV	*BMS-Slaves*
SPI	*Serial peripheral interface*
TPL	*Traffic Light Protocol*

1 Introduction

The document D4.2 presents the software architecture of the SENSIBAT BMS Master. This SW is the basis of the BMS Master that will control the SENSIBAT module composed of series connected six 5Ah – L1 cells with sensor matrices for internal pressure and temperature measurements. As a result, the spatial distribution of temperature and pressure over the cell surface are known to the BMS and can be used for advanced state estimation. The BMS collects all the information of each cell provided by the BMS slave and processes it to control the designed junction box.

The BMS-Master is composed by a Raspberry PI 4 model B, to execute the program code containing the improved SoX algorithms and to communicate with a PC or HMI, and an Arduino Due to communicate with the BMS-slave(s) and the PMDU.

2 BMS Master software design

2.1 General description

Figure 1 shows the general architecture of the hardware design, which is composed by a BMS-Master (BMS-M), a Power Measurement and Disconnection Unit (PMDU), a BMS-Slave (SLV) and a PC. On the one hand, the BMS-Master is composed by a Raspberry PI 4 model B (RPI) and an Arduino Due (ARD) and on the other hand, the BMS-Slave is based on a transceiver IC (MC33664) and an AFE IC (MC33775A), both from NXP IC. The RPI executes the program code and communicates with ARD and PC or HMI while the ARD's function is to interface with the SLV and the PMDU.

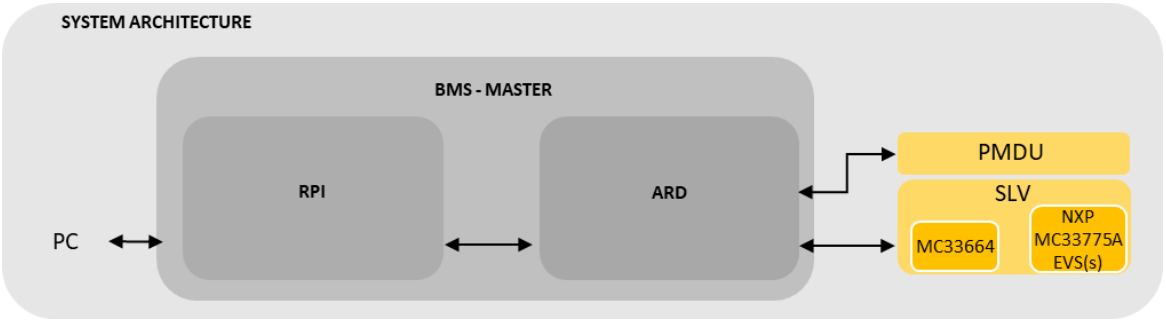

Figure 1: Battery management system architecture (hardware)

Figure 2 shows a general block diagram of the software design in addition to communication interfaces. There are two parts in the software of BMS Master: Raspberry and Arduino. As well, the PC has a software to visualise the most relevant parameters of the BMS Master and it is programmed in LabVIEW.

The RPI is programmed in the Python 3 language and is the decision-maker in the whole system. On the RPI three mains tasks are launched in parallel, the APP task is the main program cycle of BMS Master, where the algorithms will be executed, and there are two other tasks to manage the communications of the RPI. On the one hand, the RPI communicates with ARD through the COMMS_ARD task via CAN, on the other hand, the RPI communicates with the PC through the COMMS_HMI task via Ethernet or Wi-Fi.

The ARD is programmed in the C language, it activates when a request from the RPI is received. It communicates with the SLV and with the RPI. The communication with the RPI has been explained in the previous paragraph whereas with the SLV the communications are via SPI. The MC33664 is a converter that converts the SPI bus to TPL, to allow the ARD to communicate with NXP MC33775A IC. On the other hand, the ARD also controls the PMDU by analog and digital signals. In the PMDU, the digital signals can be used to activate the precharge, discharge and charge, reset hardware alarms and activate ventilation. Hardware error signals are also transmitted.

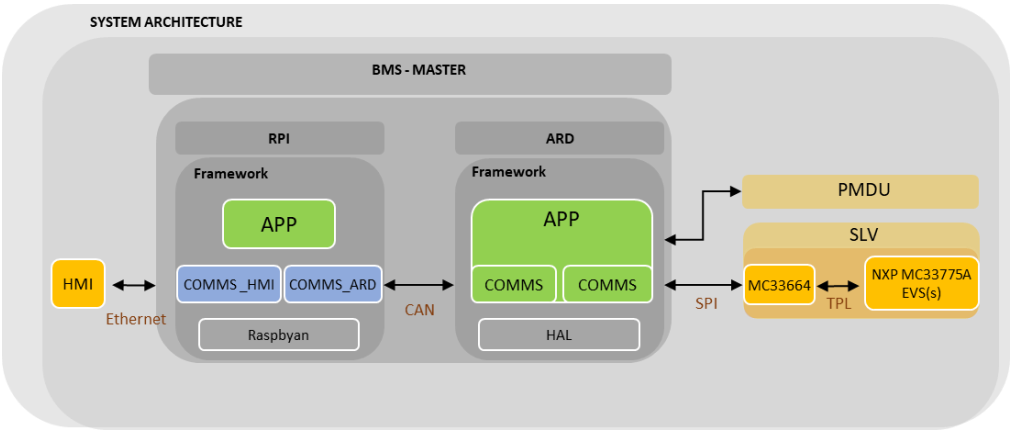

Figure 2: Battery management system architecture (software)

2.2 BMS Master SW

2.2.1 General description

Figure 3 shows a general architecture of RPI software, which is the highest level of decision making in the BMS Master and follows a multitasking philosophy. In this philosophy, three processes are running in parallel to execute different tasks on different Raspberry CPUs at the same time. The main_master is the outermost layer which creates the communications between tasks and launches multitasking. The BMS Master can self-adapt depending on the configuration, which is modified allowing it to work with or without HMI.

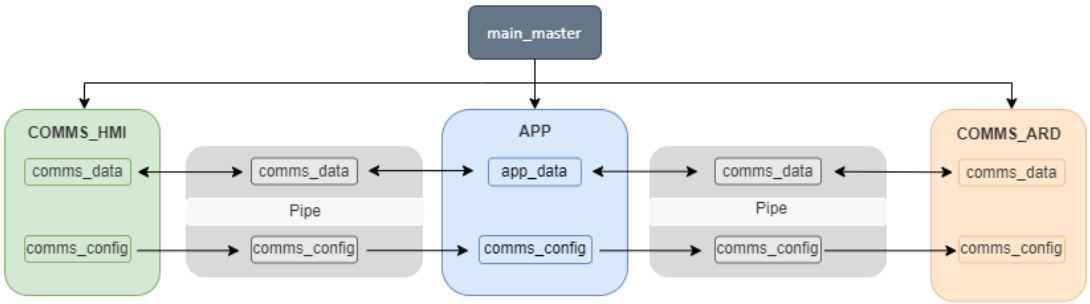

Figure 3: Architecture Software of the Raspberry

There are three main tasks, which are executed in parallel: APP, comms_HMI and comms_ARD.

- APP executes the application cycle of the BMS-Master that contains the application specific software (readings, filters, SoX algorithms, balancing, protections, state machine and writings).
- Comms_HMI executes the protocol for communication with the HMI.
- Comms_ARD executes the protocol for communication with the ARD.

In python, processes running in parallel, we choose not to share memory and variable states between processes to avoid threading problems, deadlocks, and race-conditions. To share variables between processes, python pipe classes have been used. There are two types of shared variables:

- comms_data: data variables
- comms_config: configuration parameter variables

The Figure 4 shows the blocks diagram with colors to indicate the parallel connection tasks with the application cycle. Each part of the block diagram is explained in more detail below.

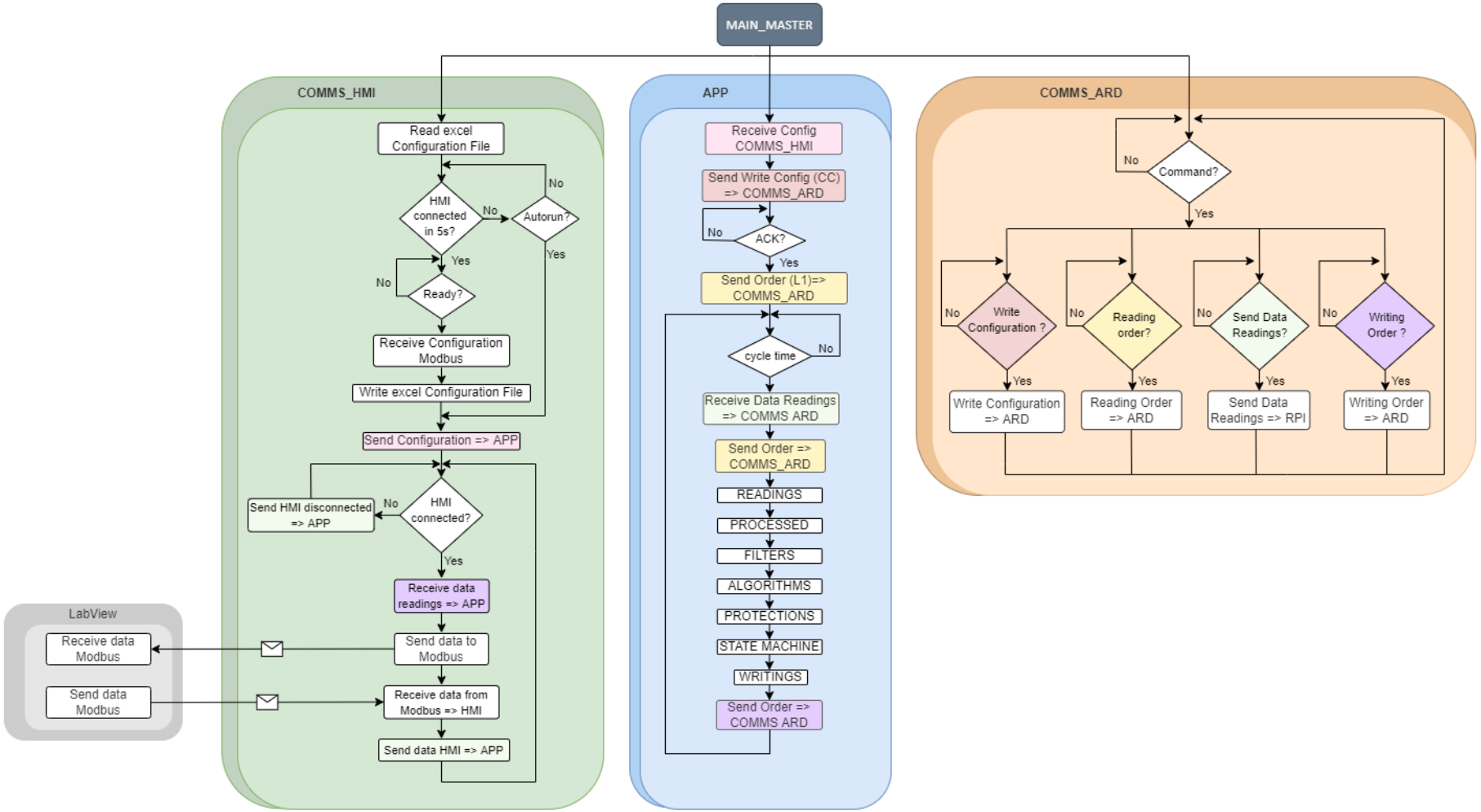

Figure 4: Cycle Diagram of Raspberry Software

2.2.2 Main SW files

2.2.2.1 APP file

2.2.2.1.1 General Description

Figure 5 shows the block diagram of the APP and its tasks can be divided in two distinct parts:

i) initialisation of the APP task
ii) repetitive cycle of the APP task

In the first part, when starting the APP task, the BMS is configured and starts the read action of the battery. In the second part, the APP cycle is executed continuously and it is the main cycle of the BMS application. Furthermore, the APP requests information and acts in all other tasks in parallel (comms_HMI and comms_ARD), which execute the necessary actions and prepare the information to the APP.

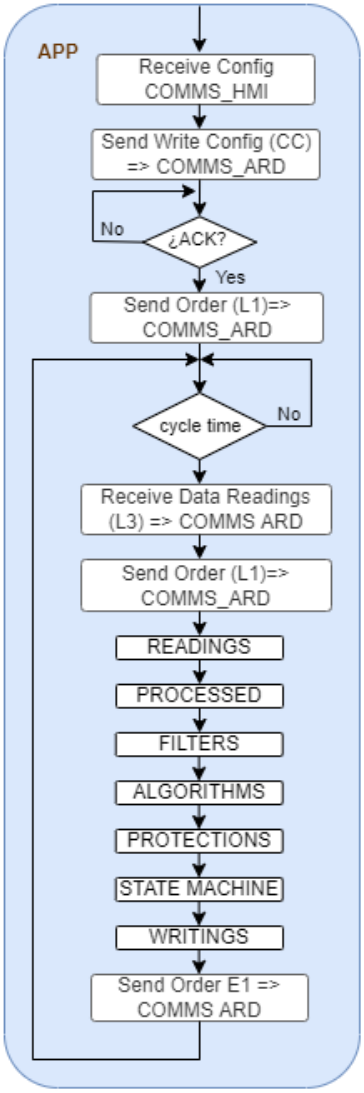

Figure 5: APP block diagram

2.2.2.1.2 Main functionality

This sub-section explains the functions shown in the block diagram in more detail.

When starting the RPI, the first thing it will receive is the configuration from the comms_HMI, then it sends the configuration to comms_ARD. In the configuration class, the cell configurations, alarm limits configurations and the rest of the configuration parameters are defined. Before starting the APP cycle, the read command is performed to make the first reading of the BMS.

After defining the configuration of the BMS, the standard cycle begins:

- Receive and save the first reads in the app_data class (from comms_HMI to APP).
- The reading order is sent to the following cycle (from comms_HMI to APP).
- The information is processed.
- The filter, algorithm and protection are applied.
- The state machine is updated.
- Writing is done in shared classes (comms_data).
- The information is transmitted by pipes to comms_HMI and comms_ARD.

This programme cycle will be constantly running as in a normal BMS mode.

2.2.2.2 Comms_ARD file

2.2.2.2.1 General Description

As already mentioned, comms_ARD is a parallel task, which as intermediary manages the communications between the APP and the ARD. Figure 6 shows the comms_ARD general block diagram, which is constantly waiting to receive a request from the APP to execute the actions that are subsequently taken. Inside of the comms_ARD task, there are four main actions of which three are from comms_ARD to ARD and only one is from comms_ARD to APP. Once it has finished performing the actions it returns to the start where it waits for the order for the next action.

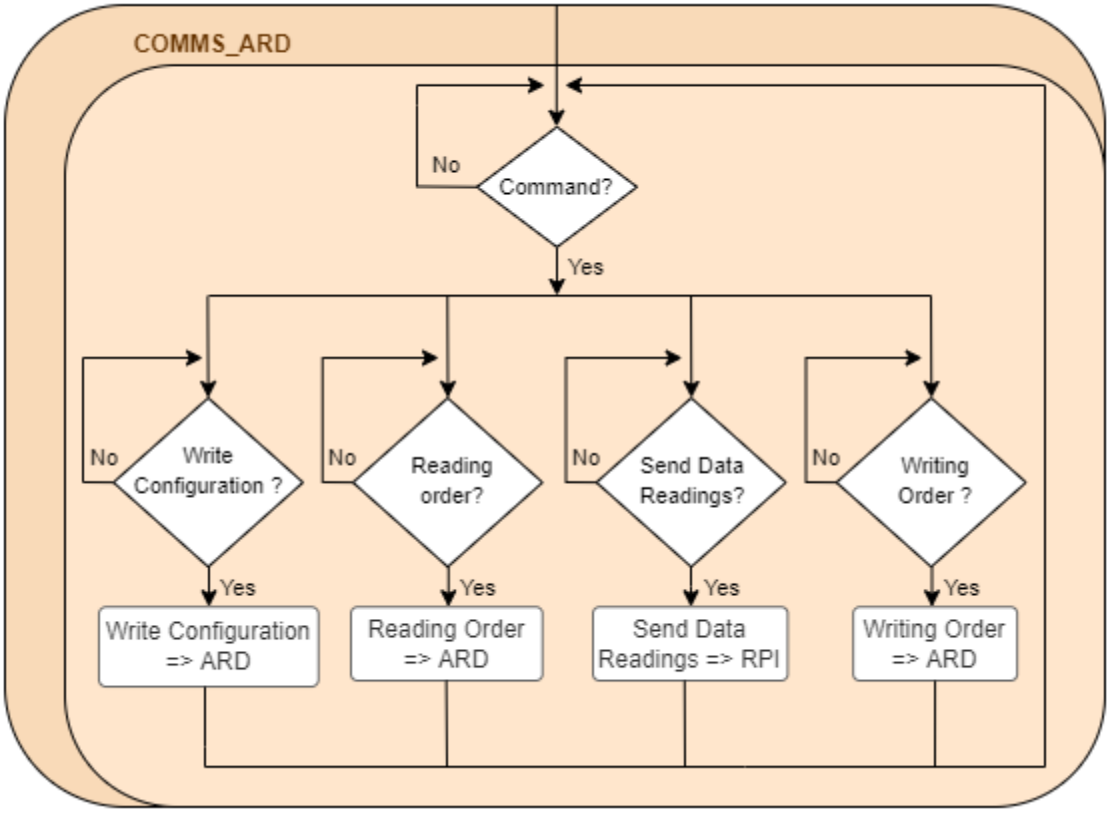

Figure 6: Comms_ARD block diagram

2.2.2.2.2 Main functionality

This sub-section explains the functions of the comms_ARD block in more detail. As can be seen in Figure 6 the comms_ARD task can receive four orders:

- Write configuration: the configuration is sent form RPI to the ARD.
- Reading order: the reading order is sent from RPI to the ARD.
- Send data readings: data reading is sent from ARD to the RPI.
- Writing order: writing order is sent from RPI to the ARD.

To send the information from RPI to ARD, the comms_ARD task has to covert the information to hexadecimal in order to transmit it by CAN protocol.

2.2.2.3 Comms_HMI file

2.2.2.3.1 General Description

As explained above, comms_HMI is a parallel task which is an intermediary that manages communications between the APP and the HMI. Figure 7 shows the block diagram of comms_HMI, it can be divided into two

distinct parts: the initialisation of the comms_HMI task, which starts the configuration of BMS Master and the repetitive cycle of comms_HMI task, which handles main communications between APP and HMI.

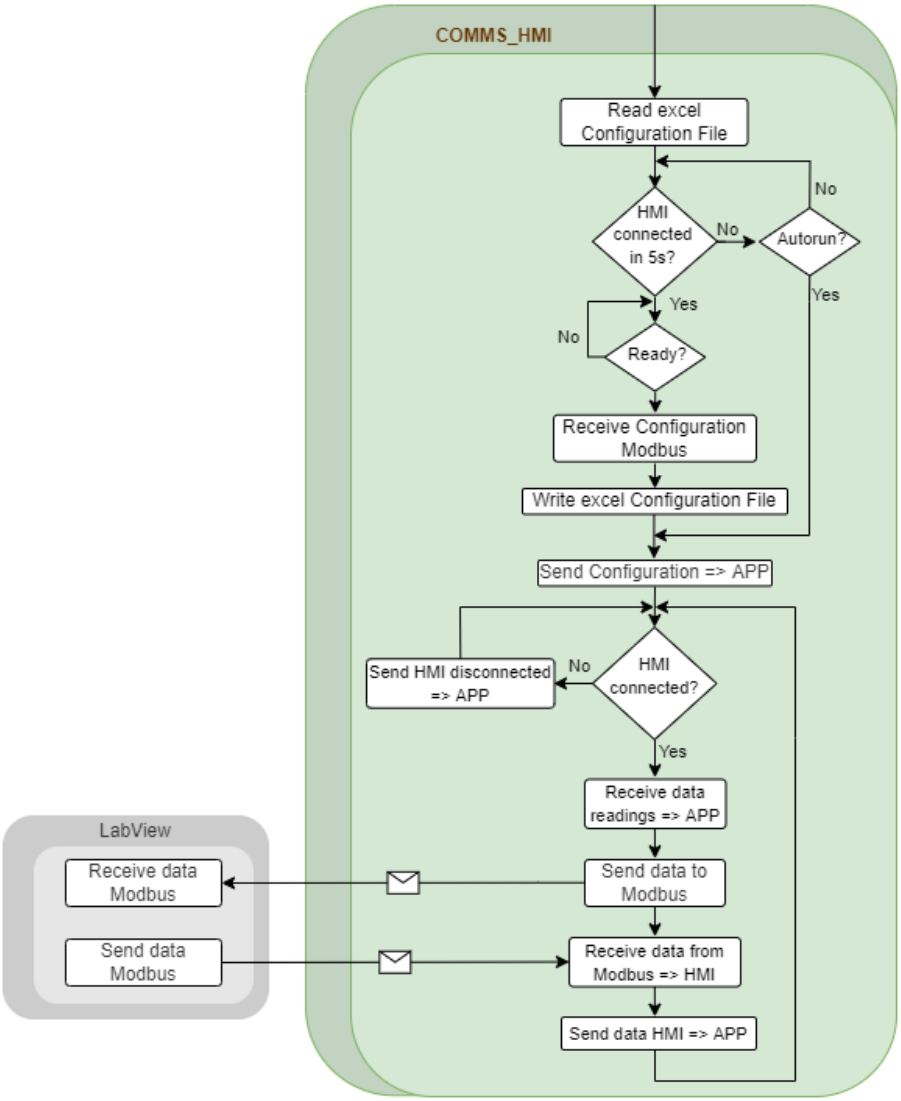

Figure 7: Comms_HMI block diagram

2.2.2.3.2 Main functionality

First, the initialisation process of the comms_HMI will be explained, in which there are two modes of operation: with HMI or without HMI, it is automatically detected. If the HMI is connected, the BMS configuration can be modified in the HMI itself and then sent to the APP. On the other hand, the autorun variable allows to run the BMS cycle without HMI, soif the HMI is not connected and **autorun** variable is **true**, the assigned configuration for the BMS will be read from the configuration excel, and if the **autorun** is **false**, neither comms_HMI nor the BMS Master will be initialised.

Once the BMS is initialised, it will enter the comms_HMI cycle in which it will check, if the HMI is connected. If this is **false**, this information will be sent to the APP. On the contrary, if the answer is **true**, the comms_HMI will receive the readings from the APP to send them by ModBus to the display interface. From this display interface it will also be possible to receive information. Once this has been received. This information will be sent to the APP and the cycle will start again from the beginning.

Figure 8 shows the graphical interface developed in LabVIEW where the most important variables of the battery are available. In addition, Figure 9 shows the 210 internal cell temperature measurements available in the BMS master. The communication between RPI and HMI have been implemented via ethernet through Modbus protocol.

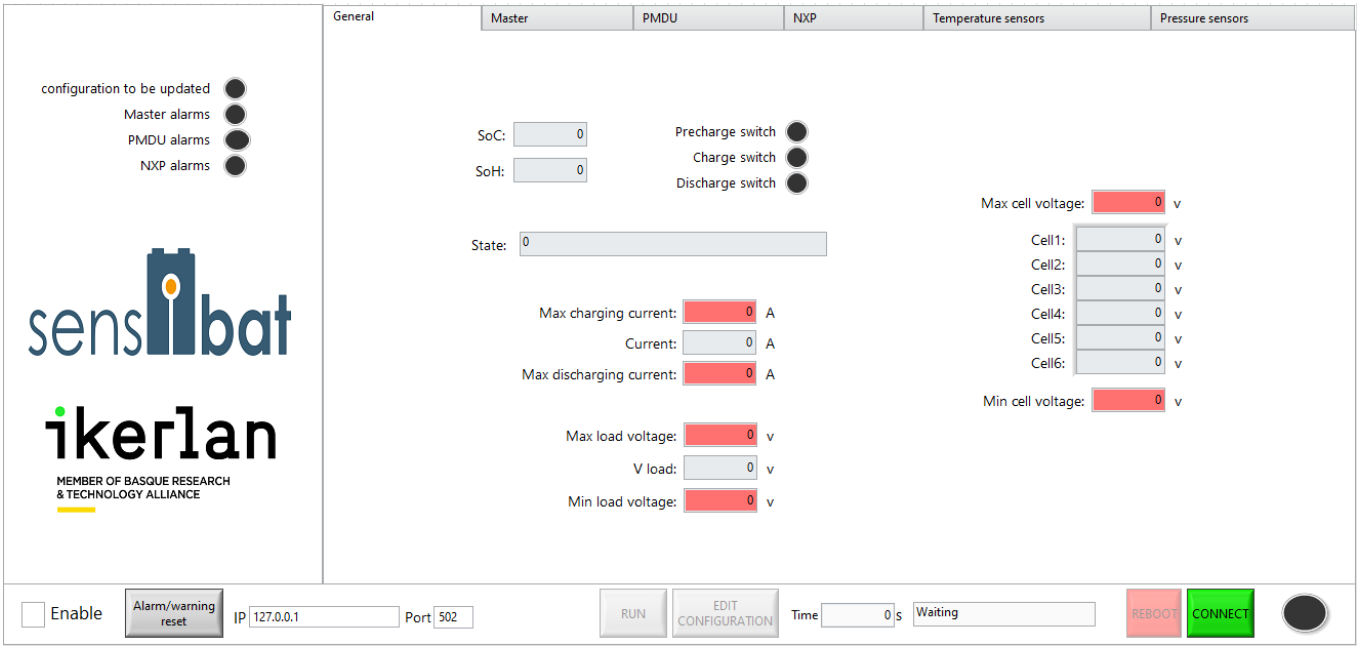

Figure 8: Graphical interface in LabVIEW

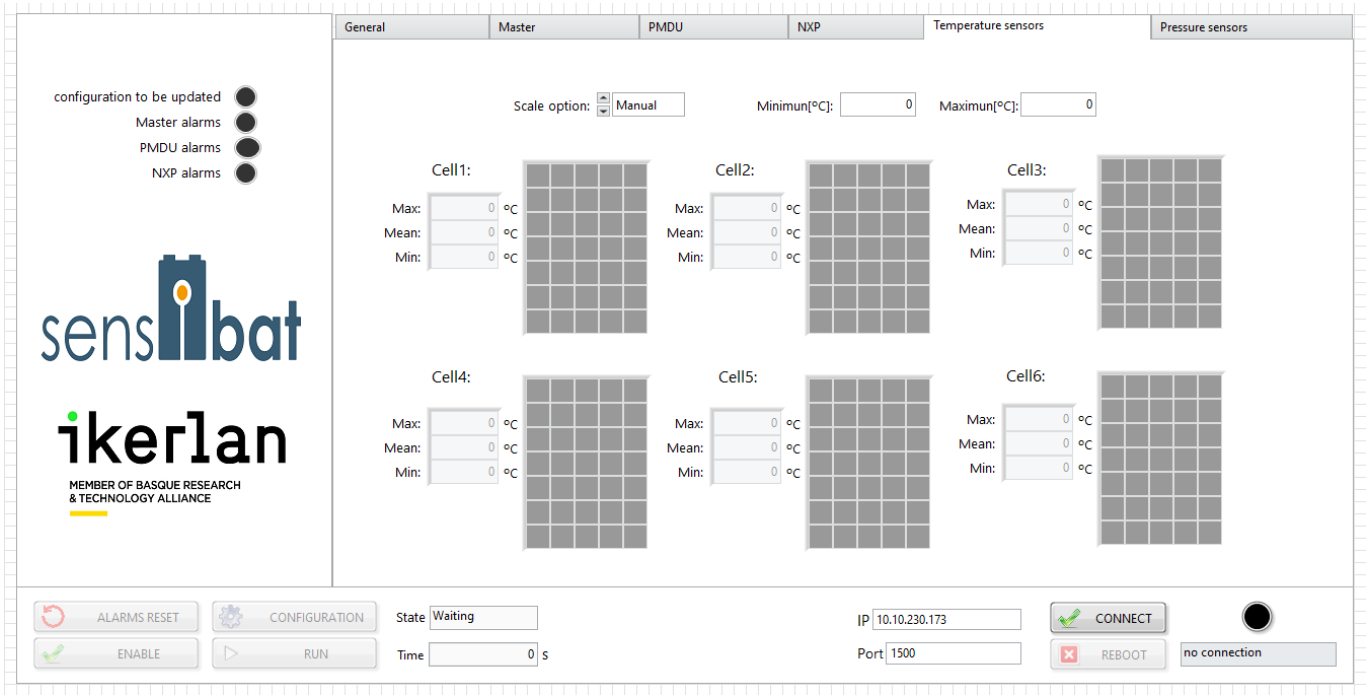

Figure 9: Graphical interface with SENSIBAT module Level 1 sensor data in LabVIEW

3 Demonstrator

Figure 10 shows a 3D visualisation of the BMS Master PCB.

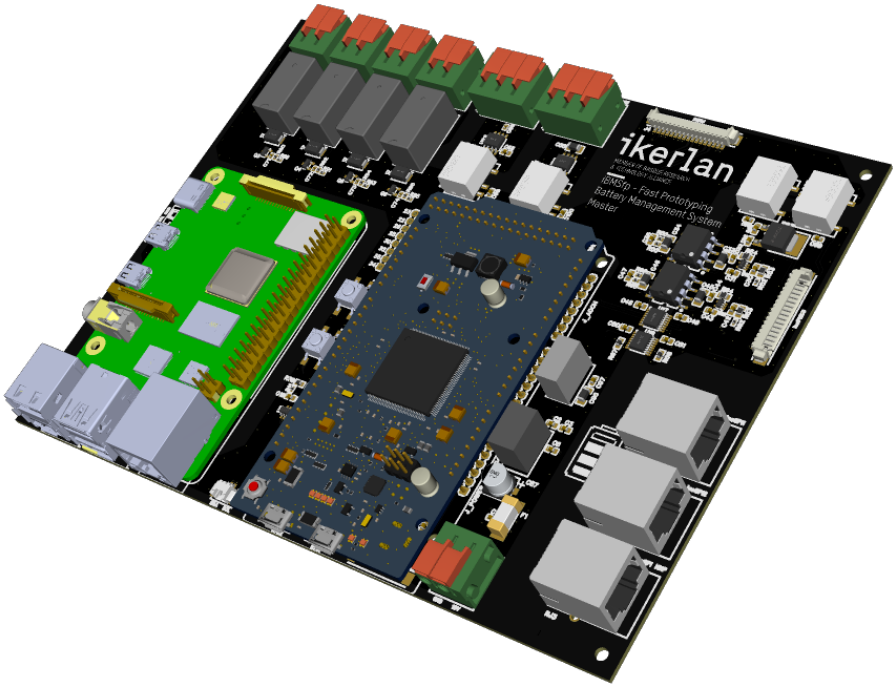

Figure 10: BMS Master PCB (rendering)

Figure 11 shows a photograph of the developed SENSIBAT BMS Master prototype based on a Raspberry PI 4 model B and an Arduino Due.

Figure 11: BMS Master prototype

4 Discussion & Conclusion

In summary, in the task 4.2 IKE developed a BMS Master based on a rapid prototyping platform. The SENSIBAT BMS master will manage the internal pressure and temperature measurements of the 24 V module provided by the BMS slave.

The present document presents a general description of the software architecture of the Master BMS composed by a Raspberry PI 4 model B and Arduino Due. This description includes the detailed software design of the Raspberry, which is the highest level of decision making in the BMS Master. The detailed description includes:

- APP executes the application cycle of the BMS-Master that contains the application specific software (readings, filters, SoX algorithms, balancing, protections, state machine and writings).

- Comms_HMI executes the protocol for communication with the HMI.

- Comms_ARD executes the protocol for communication with the ARD.

Finally, the HW design of the BMS master prototype is also presented.

5 Acknowledgement

The author(s) would like to thank the partners in the project for their valuable comments on previous drafts and for performing the review.

Project partners

#	PARTICIPANT SHORT NAME	PARTNER ORGANISATION NAME	COUNTRY
1	IKE	IKERLAN S. COOP.	Spain
2	BDM	BEDIMENSIONAL SPA	Italy
3	POL	POLITECNICO DI TORINO	Italy
4	FHG	FRAUNHOFER GESELLSCHAFT ZUR FOERDERUNG DER ANGEWANDTEN FORSCHUNG E.V.	Germany
5	FM	FLANDERS MAKE VZW	Belgium
6	TUE	TECHNISCHE UNIVERSITEIT EINDHOVEN	The Netherlands
7	NXP NL	NXP SEMICONDUCTORS NETHERLANDS BV	The Netherlands
8	NXP FR	NXP SEMICONDUCTORS FRANCE SAS	France
9	ABEE	AVESTA BATTERY & ENERGY ENGINEERING	Belgium
10	VAR	VARTA MICRO INNOVATION GMBH	Germany
11	AIT	AIT AUSTRIAN INSTITUTE OF TECHNOLOGY GMBH	Austria
12	UNR	UNIRESEARCH BV	The Netherlands

DISCLAIMER/ ACKNOWLEDGMENT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957273. The information and views set out in this publication does not necessarily reflect the official opinion of the European Commission. Neither the European Union institutions and bodies nor any person acting on their behalf, may be held responsible for the use which may be made of the information contained therein.