DEGREE PROJECT IN MECHANICAL ENGINEERING,
SECOND CYCLE, 30 CREDITS
*STOCKHOLM, SWEDEN 2021*

# Emulation of Analog Front-End isoSPI communication for Battery Management Systems


**JULIA MALACHOWSKA**

**MIKO NORE**



**KTH ROYAL INSTITUTE OF TECHNOLOGY**
**SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT**

## Authors

Julia Malachowska, julia_malachowska@hotmail.com
Miko Nore, mikonore@hotmail.com

## Examiner

Martin Törngren, Prof. Embedded Control Systems
KTH Royal Institute of Technology

## Supervisors

Muhammad Rusyadi Ramli, Ph.D.

KTH Royal Institute of Technology

Jonathan Graesser, System Test Engineer

Northvolt Battery Systems

KTH Industrial Engineering and Management

**Master of Science Thesis MMK TRITA-ITM-EX 2021:106**

**Emulation of Analog Front-End isoSPI communication for Battery Management Systems**

Julia Malachowska
Miko Nore

| Approved | Examinator | Supervisor |
|---|---|---|
| 2021-06-01 | Martin Törngren | Muhammed Rusyadi Ramli |
| | Commissioner | Contact person |
| | Northvolt Battery Systems | Jonathan Graesser |

# Abstract

This study aims to investigate how an emulator could be developed as a testing tool for Battery Management Systems (BMS) by emulating the Analog Front-end (AFE) circuit communicating with a control unit for monitoring of Lithium-Ion Batteries. All of the research was conducted in the context of the company Northvolt. By analysing data collected through a questionnaire, it was evident that an emulator testing tool could potentially make the BMS development process significantly more efficient.

A demonstrator was developed as a part of the study. It fulfilled almost all of the requirements initially stated, but required the control unit to send commands in a fixed sequence, which the current BMS control unit did not. A fixed sequence would however enable the incorporation of the developed emulator, as well as introducing other advantages such as predictability. The study showed that the most important factor to consider for developing an AFE communication emulator for BMS testing was robustness and repeatability of the timings of the communication signals.

**Keywords**: BMS, Lithium-ion battery, isoSPI, emulation, embedded system

KTH Industriell teknik och management

**Examensarbete MMK TRITA-ITM-EX 2021:106**

**Emulering av analog front-end**
**isoSPI-kommunikation för batteristyrsystem**

Julia Malachowska
Miko Nore

| Godkänt | Examinator | Handledare |
|---|---|---|
| 2021-06-01 | Martin Törngren | Muhammed Rusyadi Ramli |
| | Uppdragsgivare | Kontaktperson |
| | Northvolt Battery Systems | Jonathan Graesser |

# Sammanfattning

Syftet med denna studie var att undersöka hur ett testverktyg baserat på en emulator skulle kunna utvecklas för batteristyrsystem. Studien genomfördes på batteriföretaget Northvolt. Genom att analysera data insamlad via ett frågeformulär framgick det tydligt att ett testverktyg baserat på en emulator hade god potential att göra utvecklingsprocessen av batteristyrsystem mer effektiv.

En prototyp utvecklades som en del av studien. Denna uppfyllde nästan alla de initialt uppsatta kraven, men var anpassad för kommunikation i en fix sekvens, till skillnad från det aktuella systemet hos företaget. Via studien fann man att implementationen av en fix kommunikationssekvens skulle medföra önskvärda egenskaper hos systemet såsom förutsägbarhet. Vidare visade studien att den viktigaste faktorn att ta i beaktning för utveckling av en emulator var robusthet och repeterbarhet hos timingen av kommunikationssignalerna. Detta eftersom kommunkationen mellan enheter förlitar sig på korrekt timing av varje skickad bit för korrekt inlästa meddelanden.

**Nyckelord**: Batteristyrsystem, litiumjonbatteri, isoSPI, emulering, inbyggda system

# Acknowledgements

First and foremost, we would like to thank our industrial supervisor Jonathan Graesser, who have motivated us throughout the project and led us in the right direction. We would also like to thank the entire Northvolt Battery Systems team and especially Simon Nelson Landén for support and engagement in the project.

We would also like to thank our supervisor at KTH, Muhammad Rusyadi Ramli, who have been very encouraging from the start and provided well-needed guidance. Thank you also Martin Törngren, our examiner, and Fredrik Asplund, course leader, who have guided us and provided valuable feedback.

Lastly, a huge thank you goes to Jan Kamidra from the Arm Mbed OS support forum who put time and effort into supporting our trouble-shooting when things did not go as planned. We are forever grateful that you took the time.

Miko and Julia
Stockholm, May, 2021

# Contents

# List of Figures

# List of Tables

# List of Acronyms

**ADC** Analog-to-Digital Converter. 20, 38

**AFE** Analog Front-End. 4, 13, 42, 45

**BMS** Battery Management System. vii, xii, 1–8, 10–14, 18–21, 23, 25, 29, 34, 42, 43, 45–48

**CAN** Controller Area Network. 8, 9

**CS** Chip Select. 8, 13, 32, 36

**CSV** Comma-Separated Values. 14–16, 22

**DMA** Direct Memory Access. 29, 44

**DR** Design Requirement. 25, 31, 35, 36

**EV** Electric Vehicles. 1, 2, 4, 7

**FR** Functional Requirement. 35, 36, 44

**GPIO** General-Purpose Input/Output. 20

**HAL** Hardware Abstraction Layer. 29, 30, 44, 49

**HIL** Hardware-In-the-Loop. 3, 10, 11, 39, 40, 43, 46, 47

**HLF** High-level factor. 37, 38, 42, 43

**$I^2C$** Inter-Integrated Circuit. 8, 9

**IC** Integrated Circuit. 8, 23, 24, 34, 39, 49

**isoSPI** Isolated Serial Peripheral Interface. vii, 2–4, 8, 9, 13, 17–20, 24–26, 34, 42

**LIB** Lithium-Ion Battery. 1, 2, 6, 10, 12, 19, 47

**LLF** Low-level factor. 37–39, 43

**MCU** Microcontroller Unit. vii, 3, 8, 17, 19, 25, 44, 47–49

**MISO** Master-In-Slave-Out. 8, 13, 14, 22, 36

**MOSI** Master-Out-Slave-In. 8, 13, 14, 21, 36

**MUX** Multiplexer. 20

**PEC** Packet Error Code. 15, 17, 21, 22, 24, 26, 29, 35, 36, 38, 43

**PLADC** Poll ADC Converter Status. 24

**QR** Quality Requirement. 35, 36, 44

**RDAUX** Read Auxiliary Voltages. 24

**RDCFG** Read Configuration. 24, 34

**RDCV** Read Cell Voltages. 16, 24

**RDSTAT** Read Status. 24, 34

**RQ** Research Questions. 3, 4, 12–15, 22, 42, 44, 45

**RS-232** Recommended Standard 232. 8, 9

**RS-485** Recommended Standard 485. 8, 9

**SCLK** Serial Clock. 8, 13, 36, 44

**SOA** Safe Operating Area. 2, 10

**SOC** State Of Charge. 2, 6, 41

**SOH** State Of Health. 2, 6, 41

**SPI** Serial Peripheral Interface. 8, 9, 13, 14, 17, 24, 26, 28–30, 35, 44, 45

**UART** Universal Asynchronous Receiver-Transmitter. 8, 9

# Glossary

**AFE IC** Analog Front-End Integrated Circuit, a circuit in direct connection with cells of a battery pack for measuring cell and thermistor voltages. vii, xii, 2–4, 13–15, 18, 20, 21, 24, 30, 34, 35, 43

**BMU** Battery Management Unit, the main BMS processor/control unit running the BMS software. vii, 2, 3, 12–15, 19–21, 23, 25, 34–36, 44, 45, 48

**CSB** Cell Sensor Board, a printed circuit board that integrates multiple AFE ICs. vii, 2, 3, 10, 12–15, 19–22, 25, 34–36, 38, 41, 45

# Chapter 1

# Introduction

This chapter provides the reader with an introduction to the thesis project. The project background is presented, followed by descriptions of the problem, objective and scope, as well as a clarification of certain important terminology.

## 1.1 Background

The industry, and our society as a whole, is moving towards electrification, and rechargeable batteries are a fast-emerging technology for energy storage as the global demand for power is increasing [1, 2]. Lithium-Ion Battery (LIB) chemistries in large part outrival other battery chemistry options because of their high energy and power density. There are, however, challenges associated with developing LIB technology. Battery packs used for high-voltage applications, such as Electric Vehicles (EV), constitute serious safety risks if not handled correctly. An improper handling may also lead to a significant shortening of the battery lifetime and decreased performance. These are the main reasons why Battery Management Systems (BMSs) have been developed in parallel with the actual batteries.

A BMS is a system built around an electric device that manages a rechargeable battery pack, maintaining its safety, longevity and performance. In the case of LIBs, the BMS is in charge of preventing voltages, temperatures and currents from exceeding certain upper and lower limits. Additionally, voltage balancing amongst cells is a crucial part of a BMS, as it significantly increases the battery performance and longevity [3]. Furthermore, a more advanced BMS does more than balancing and keeping the battery

within a Safe Operating Area (SOA). It has a broad set of features optimising the battery performance and longevity, and manages e.g. temperature control systems and fault tolerance routines [4]. Based on the cell data voltage, current and temperature, the BMS may also estimate and act on useful information such as the battery State Of Charge (SOC)[1] and State Of Health (SOH)[2] [3].

A BMS is a complex system integrating many hardware and software components, and depending on the battery manufacturer it can be implemented differently. A common BMS architecture is illustrated in Figure 1.1.1. It has Analog Front-End Integrated Circuits (AFE ICs), integrated onto circuits called Cell Sensor Boards (CSB), in direct connection with the battery cells, separate from a main BMS processor/control unit, called Battery Management Unit (BMU). With this architecture, a communication link is required between the BMU and the CSBs. Isolated Serial Peripheral Interface (IsoSPI) is one communication interface option that has proven to be particularly beneficial for high-voltage batteries operating in electrically noisy environments such as EVs [5].

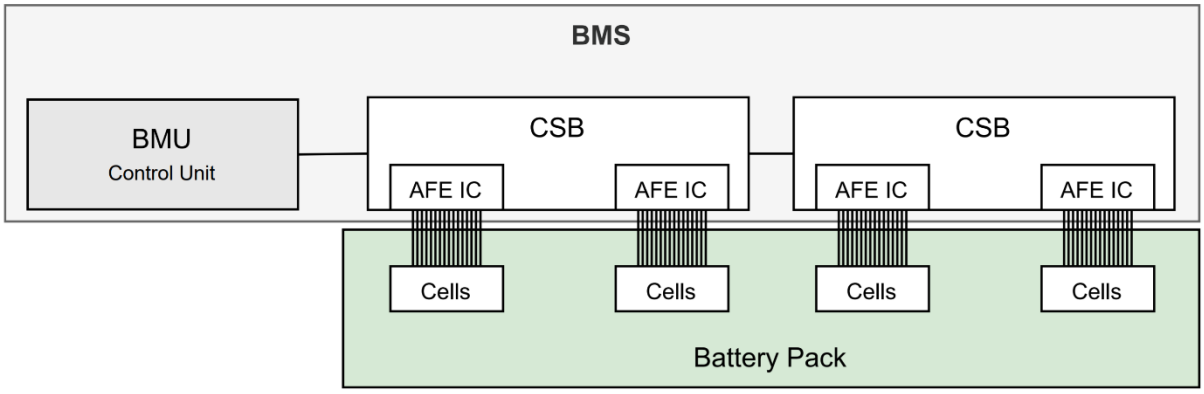



Figure 1.1.1: A BMS architecture with two CSBs and four AFE ICs connected to cells.

## 1.2 Problematization

The Battery Systems department at Northvolt, a Swedish battery developer and manufacturer specialising in LIB technology, regularly executes extensive testing on the company's BMS hardware and software, as required for development and quality assurance. Today, testing of the BMS is dependent on large hardware setups, as it is

[1]SOC is the proportion of charge available at a certain point in time compared to the total charge available when fully charged.

[2]SOH is a measure of the overall condition of the battery compared to nominal conditions.

connected to an actual battery pack. Nevertheless, for a significant part of the BMS testing process, the full complexity of the behaviour of a real battery is not required. Therefore, battery cell simulators integrated in a Hardware-In-the-Loop (HIL) rig are also used to some extent. However, commercial HIL rigs are also relatively bulky, hence not scalable and with the biggest drawback being their high cost. A more flexible test rig, based on emulation[3] of the CSBs and cell data, could mean making the testing process significantly more efficient.

The BMS architecture used at Northvolt today is the aforementioned, where CSBs are separate from a BMU. The AFE IC here is a cell monitor that has a built-in isoSPI interface. A flexible, scalable and affordable test rig for BMS testing at Northvolt would have to emulate the AFE IC functionality as well as cell data, and have an isoSPI communication interface. Since a BMS is a complex system operating in real time, creating such an emulator requires taking many factors into consideration.

## 1.3 Objective and Research Question

The aim of the study is to investigate how AFE ICs communicating through an isoSPI interface can be emulated towards a BMU, as if there are real battery cells connected, for the BMS architecture presented in Section 1.1. Furthermore, the study aims to evaluate the potential impact of a test rig based on such an emulator.

Figure 1.3.1 shows a schematic overview of a BMU communicating with an AFE IC emulator in a master-slave configuration. In the figure, the emulator, acting as communication slave, contains a Microcontroller Unit (MCU) running the emulation software of one or more AFE ICs and associated cell data. The development of an emulator as part of the research strategy of this thesis is described in Chapter 3.

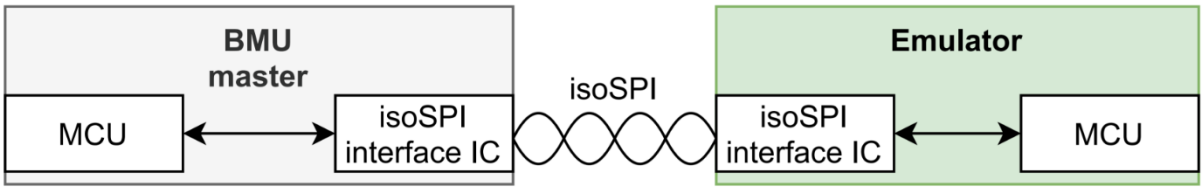



Figure 1.3.1: An AFE IC emulator running on an MCU, communicating cell data to a BMU through an isoSPI interface.

The two Research Questions (RQ) to be answered are:

[3]For a definition of *emulation*, see Section 1.5

**RQ1:** What are the factors to consider when creating an Analog Front-End (AFE) isoSPI communication emulator for a highly accurate BMS like Northvolt's BMS?
**RQ2:** Considering differences in the quality and functionality of the emulation of the factors identified in RQ1, what is the impact on time, quality and cost by introducing the aforementioned emulator test rig for different engineering disciplines?

## 1.4 Delimitations

In order to answer the RQs, a requisite BMS implementing the isoSPI protocol and AFE ICs was investigated. The BMS currently developed at Northvolt fulfills these requirements and in addition, is optimized for functional safety and accuracy, crucial for the applications such as EVs and Energy Storage Systems that have high management requirements. The study is therefore delimited to a particular BMS implementation. Consequently, the system characteristics of a BMS based on design choices, such as functionality and architecture, are fixed. The analysis induced by RQ2, namely the impact of an emulator test rig, will also be made in the context of Northvolt Battery Systems. Since all of the software and hardware associated to the studied BMS is confidential property of the company, no details thereof are enclosed in this thesis.

## 1.5 Terminology

There are no unambiguous definitions of the concepts emulation and simulation. Some sources even suggest definitions which other sources exchange between the two terms. Since the concept *emulation* will be frequently used in the thesis, a description of what definition is used is on its place:

> *"Emulation is the process of imitating a hardware/software program/platform on another program or platform. [...] Emulators, as the name implies, emulate the functions of one system on another. Thus, the second system behaves like the original system, attempting to exactly reproduce the external behaviors of the first system." [6]* [4]

[4] In contrast, the same source gives the following definition of *simulation*: "A simulation is any research or development project where researchers or developers create a model of some authentic phenomenon. Many aspects of the natural world can be transformed into mathematical models, and using simulation allows IT systems to mimic the outcomes that happen in the natural world." [6]

# Chapter 2

# Battery Management Systems

This chapter provides the reader with a theoretical background as it presents an overview of BMS characteristics on a system level as well as an overview of related work on emulation for BMS testing.

## 2.1 System characteristics

BMS is a broad term that covers a variety of systems with the common goal to manage a battery in some way. The book *Battery Management Systems for Large Lithium-Ion Battery Packs* defines it as ” [...] any product or technology used with the intent of taking care of a battery in one way or another” [3]. Furthermore, the author categorizes different BMSs with respect to its functions. The article *Decentralized Master-Slave Communication and Control Architecture of a Battery Swapping Station* brings up system architecture and topology as factors to categorize BMSs. Communication is also a relevant factor for BMS design [7].

A general high-level representation of a BMS is presented as a schematic in Figure 2.1.1. The functionality and architecture of the BMS are illustrated as named boxes with connections to each other. The following sections will explain these BMS characteristics deeper. As shown in the figure, the BMS is integrated between a battery pack and an external system.

Since this thesis is focused on the BMS on a system level, the details of electrical implementations is not considered relevant and will not be covered in the following sections.

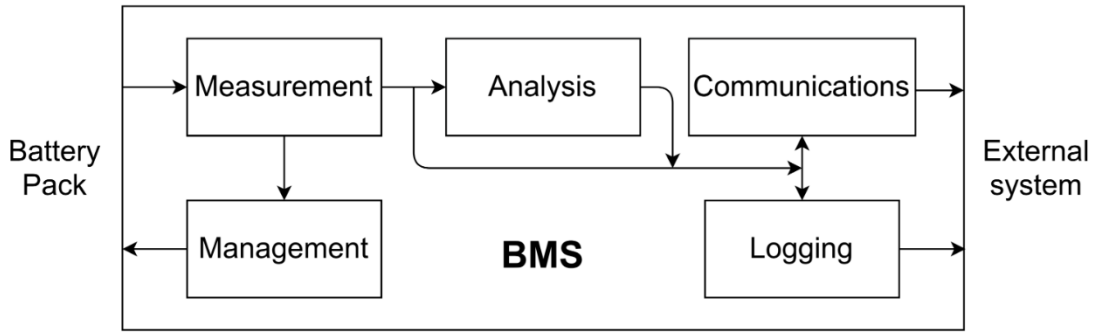



Figure 2.1.1: A general BMS on system level.

### 2.1.1 Functionality

Different LIB applications require different levels of complexity of its BMS. Therefore, BMS functions can range from just monitoring the battery, to protecting it, or even performing state estimation, maximizing its performance and reporting to users and/or external devices [3]. The typical BMS gathers data of:

- Voltage - both of individual cells and the pack
- Temperature - either of cell, battery or suitable points
- Current - to and from the battery pack [3, 7]

This is represented by the box *Measurements* in Figure 2.1.1. As touched upon in Section 1.1, the functions of a BMS, beyond gathering data, may also include:

- Battery protection, comparing the cell data with upper and lower limits
- Control of the charging and discharging
- Balancing of the voltage amongst cells (a function called cell balancing)
- Estimating SOC, on a scale from 0 to 100 %
- Estimating SOH, as both calendar and cycling ages degrade the battery over time
- Thermal management
- Internal and external communication [7]

The box *Management* in Figure 2.1.1 lumps the control of charge to and from the battery as well as thermal management. The *Analysis* box represents SOC and SOH algorithms, and *Logging* stands for the BMS's possibility to store all its processed data.

### 2.1.2 Architecture

Different architectures impact how the electronic circuits and the functions are separated, as well as the hardware and software functions distribution and validation procedures [8]. There are three main BMS architectures, namely centralized, distributed and modular. A centralized BMS has one central control unit in control of all of the cells of a pack, in contrast to a distributed BMS, where each cell is connected to its own controller, which in turn communicates with a central controller. The modular type of BMS architecture can be considered a combination between centralized and distributed. For such an architecture, a group of cells of a battery pack is controlled by the same unit. These units communicate with a central controller via a serial communication interface. The most common approach regarding battery packs for EVs is a modular architecture with master-slave configuration. The master-slave configuration can further be divided into different topologies, that describe how they are connected physically. These are: star, bus, ring/daisy chain and hybrid [7]. Figure 2.1.2 illustrates the different kinds of basic topologies.

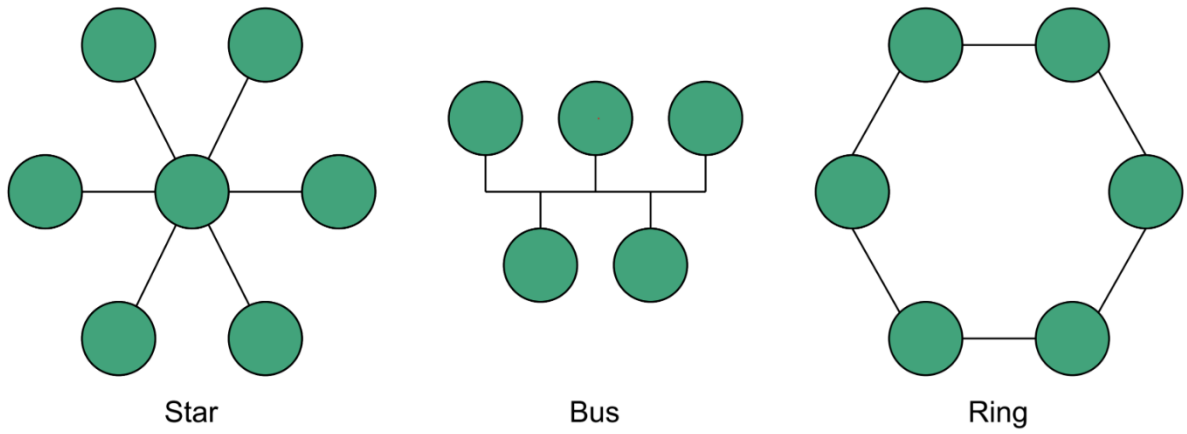



Figure 2.1.2: Schematics of different master-slave configuration topologies.

Referring to the architecture of a general BMS, as presented in Figure 2.1.1, the focus of this thesis is the box *Measurement* in connection with *Management*.

### 2.1.3 Communication

There are numerous communication links in a BMS, internally and externally. Since the external communication is outside of the scope of this study, only internal communication will be reviewed here. The internal communication can further be categorized based on the requirements, where the distance between devices is an important factor. Generally, communication taking place on a circuit board, such as between Integrated Circuits (ICs) can be considered short distance, and between different circuit boards can be considered long distance [9, 10].

Communication protocols that are used for short distance are typically: Inter-Integrated Circuit ($I^2C$), Serial Peripheral Interface (SPI) and Universal Asynchronous Receiver-Transmitter (UART), whereas common protocols for long distance are: Controller Area Network (CAN), Recommended Standard 232 (RS-232) and Recommended Standard 485 (RS-485). These communication protocols have been used for decades and are well established in the industry [7, 11, 12].

Analog Devices launched a new type of interface in 2016 called isoSPI that has many advantages, competing with the more traditional protocols. IsoSPI operates with regular SPI at both ends of the communication lines and thus, making it compatible with the regular four SPI pins of MCUs: Chip Select (CS), Master-Out-Slave-In (MOSI), Master-In-Slave-Out (MISO) and Serial Clock (SCLK) [5]. Advantages and disadvantages of the different protocols are summarized in Table 2.1.1 and 2.1.2.

Presented in Table 2.1.1 are $I^2C$, SPI and UART. They all have advantages and disadvantages, making none of them superior in the general case. The choice between them depends on what features are considered favorable for one particular application. The same goes for the traditional long distance protocols compared in Table 2.1.2. IsoSPI stands out, as the only disadvantage is that it lacks built-in error handling. It does not necessarily pose a problem since error handling can be implemented in other ways.

Table 2.1.1: Pros and cons of different communication protocols for short distances.

| Protocol | Advantages | Disadvantages |
|---|---|---|
| **$I^2C$** | Requires only two wires [13] | Limited range of speeds [14] |
| | Capable of handling multiple masters and multiple slaves on the same bus [13] | Consumes more power than SPI and UART [15] |
| | Enables error handling with acknowledgements [15] | High overhead with addressing and acknowledgements [14] |
| **SPI** | Data can be streamed continuously without interruption [14] | 4 wires required [15] |
| | Full duplex, faster than $I^2C$ [15] | No error handling [15] |
| | High data transfer rate [15] | |
| **UART** | Requires only two wires [15] | Handles communication between two devices only [16] |
| | Enables error handling with parity bit [15] | |
| | Full duplex [17] | |

Table 2.1.2: Pros and cons of different communication protocols for long distances.

| Protocol | Advantages | Disadvantages |
|---|---|---|
| **CAN** | Robustness to noise [18] | Half duplex [17] |
| | Enables multiple ways of error detection [18] | High software expenditure [19] |
| **RS-232** | Requires only 2 wires [20] | Relatively slow [21] |
| | Enables error detection with parity bit [22] | Handles communication between two devices only [21] |
| **RS-485** | High noise immunity [23] | 4 wires required [20] |
| | High speed [23] | Requires costly wires [24] |
| **isoSPI** | High noise immunity [5] | No error handling [5] |
| | Requires only 2 wires [5] | |

## 2.2 Emulation for BMS testing

In order to validate the full range of functionalities of a BMS, it must be submitted to different scenarios during testing. Generating full coverage test cases with physical batteries is difficult, as it is time-consuming and dangerous to replicate each case. The system's reaction at the limits of, and beyond, the SOA of the battery and cells need to be tested, which is not possible with real cells. It is also costly to produce in scale as it would require a large number of prototypes. Emulation and simulation for testing purposes, integrated onto HIL test rigs, is one solution to the issues that come with testing with the real hardware. It is a widespread procedure within the industry, and is known to bring along reductions of costs, resources and development time [25]. In addition to reductions of costs, resources and time, introducing HIL techniques to BMS testing can also improve the development process in terms of flexibility, traceability, safety and reproducibility. However, commercial HIL solutions on the market today are expensive and not well suited for small volume prototypes. As described in Section 1.1, BMSs of different battery system producers have different characteristics and thus require different, tailored HIL rig layouts. There is no standard way of implementing HIL into a testing process and many researchers and developers, motivated by the aforementioned advantages of HIL, have developed and reviewed cell and battery emulators in parallel, examples are [4, 8, 26, 27].

The researchers and authors of [4] developed an advanced HIL test rig for BMS testing. A LIB model programmed in C code was integrated in a system-level simulation and run in real time. The capabilities of the setup were demonstrated through experiments with BMS test cases. The developed HIL simulation consisted not only of the battery model, but also of model management controllers and other models of typical components of a LIB system, such as temperature and current sensors, all for a BMS using master-slave configuration. Each slave unit could be configured to supervise eight cells and two temperatures. Four cell monitoring slave circuits, or CSBs, were used. It was stated that in order to enable a universal BMS validation, a library of battery models for different cell sizes, formats, chemistries, and manufacturers would be desired, and also that such a library is not found in the literature. The conclusion of the research was that in comparison with tests done on real batteries, BMS validation tests conducted on the developed platform may be more cost and time effective, easier to reproduce, and safer beyond the normal range of operation, especially at early stages

in the development process or during fault simulation. It should be noted that the BMS under test in this research is built according to customer-specific requirements and hence the generalizability of the findings can be discussed.

In the research presented in [8], a modular, affordable and flexible multi-cell emulator was designed for testing and validation of BMS functionalities. The emulator could either be cell-based or communication-based, depending on the test objectives, the functions to be validated, the topology and the level of integration of the design. A cell-based approach could provide all the simulated signals needed at a cell level and was suitable to test a larger variety of BMSs and functions. A communication-based approach was used for modular architectures with master-slave configurations, where sensors were emulated through the communication bus of the system.

In [8], the results showed that a HIL rig reduces the validation testing time in comparison with tests conducted with real battery cells. The advantages included increased efficiency, replicability, safety and flexibility as it allowed testing beyond normal use cases. The emulator was able to show good performance compared to a real battery cell, while being an affordable solution for simulation test systems.

# Chapter 3

# Method

This chapter will give the reader a description of, and reasoning for, the research strategy that was applied in the project, including methods for data collection and data analysis. Additionally, an evaluation is made on the research validity.

## 3.1 Research Strategy

The research presented in this thesis applied an exploratory research approach through a case study. According to [28], a case study is as a strategy involving *an empirical investigation of a particular phenomenon using multiple sources of evidence*. Since the objective of the research was to investigate how CSBs can be emulated towards a BMU, as described in Section 1.3, it was considered necessary to understand how the system was designed and how the communication between the two units worked. A case study was therefore done, as it could provide the necessary qualitative data to answer RQ1. The investigated case was the development of an emulator for the BMS developed at the company Northvolt. Northvolt has the ambition to meet the increasing demand for LIBs of several industries for decades to come. As described in Section 1.1, a LIB meeting high requirements on performance and safety relies on a BMS, and with Northvolt's ambition and success hitherto, their BMS was considered suitable for the research. Regarding RQ2, with the aim to evaluate the potential impact of an emulator test rig for BMSs, it was considered relevant to collect data within teams working with BMS testing, elaborated in the following sections.

## 3.2 Data collection

#### Research Question 1

*What are the factors to consider when creating an AFE isoSPI communication emulator for a highly accurate BMS like Northvolt's BMS?*

In order to research what factors to consider when creating an AFE IC isoSPI communication emulator for a highly accurate BMS, data was initially collected through observation. This was done partially by sampling data from the internal communication of the case BMS, which generated quantitative data, as well as by studying datasheets of the devices communicating within the case BMS, which can be considered qualitative data. More specifically, the quantitative data that was sampled was a data log from the communication data between the BMU and the CSB. The data logs were essential for understanding the communication that would be emulated in order to answer the RQ.

In addition to the observed data, an emulator demonstrator was developed. The development of a demonstrator emulator generated both quantitative and qualitative data. The quantitative data generated by the demonstrator development was similar to the quantitative data gathered by observation of the BMS. Namely, samples of communication between the emulator and other devices in stages when the emulator was ready to be used. The qualitative data was insights generated throughout the development process itself.

The logic analyzer *SP209* from Ikalogic was used in order to sample a log from the communication taking place between the BMU and CSB. The 9-channel logic analyzer enabled 8-bit parallel data to be captured along with a clock signal with 200MHz timing resolution [29]. Figure 3.2.1 shows how the logic analyzer was connected to the BMS when capturing the SPI communication signals at the master end of the isoSPI communication. The four SPI channels, CS, MOSI, MISO and SCLK, were each connected to the logic analyzer. The *SP209* logic analyzer relies on the software ScanaStudio in order to capture, display and decode signals [30]. Using the SPI protocol decoder available in ScanaStudio, the logic changes in the MOSI and MISO channels could be converted from single bits to hexadecimal numbers with one byte of information each.

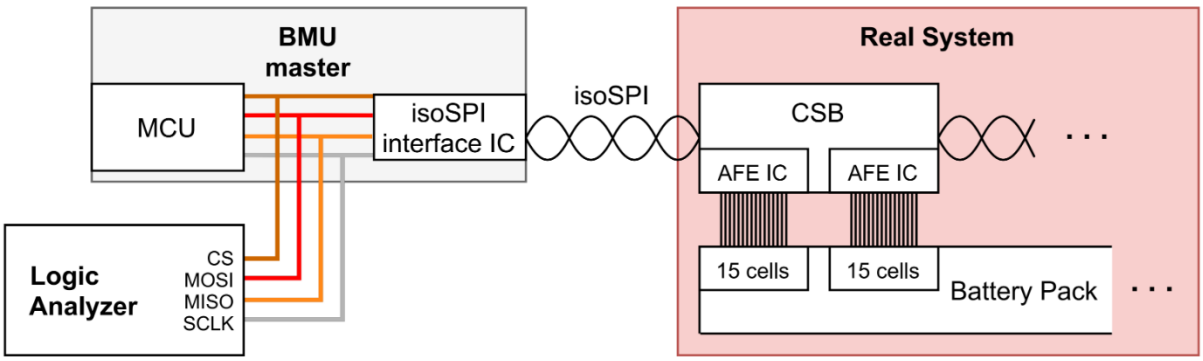



Figure 3.2.1: A logic analyzer connected to a BMU communicating with a CSB.

The structure of the communication was such that the BMU would command the CSB to send some data stored in the memory of the AFE ICs, to which the CSB would respond with the requested data. The communication was logged for 50s, corresponding to over 100,000 samples, enough to include many repetitions of all the command and response sequences that were considered fundamental for an emulator that could potentially replace the *Real System* block in Figure 3.2.1. The logged and decoded-into-hexadecimal SPI signals from the MOSI and MISO channels were saved as a Comma-Separated Values (CSV) file, exported from the ScanaStudio software.

### Research Question 2

*Considering differences in the quality and functionality of the emulation of the factors identified in RQ1, what is the impact on time, quality and cost by introducing the aforementioned emulator test rig for different engineering disciplines?*

In order to answer RQ2, a structured questionnaire was designed and answers were collected from a group of engineers with experience from BMS testing at the case company. Since RQ2 is concerned with the effects of an emulator test rig, this was considered an adequate method to provide the insight needed to answer the RQ.

The target group was selected based on how involved the engineers were with the BMS testing in their regular work. The more the engineer was involved made them better suited for participating. The questionnaire was distributed to the target group consisting of 26 employees via messages at the company's online platform and they were given 10 days to submit their answers, resulting in a 96.2% response rate.

The questions were designed such that the answers would enable answering RQ2 and thus addressed time, quality and cost. It consisted of multiple-choice questions and open-ended questions. The questionnaire also addressed RQ1 by questions that were measured on a 5-point Likert scale, where the respondents could rate the importance of different emulator features. The questionnaire along with the information that the respondents were provided with prior to answering can be viewed in Appendix B.

### 3.2.1 Data Collection Summary

For RQ1, quantitative data was collected as communication data logs, both of the real system and of a system with the emulator. Qualitative data for RQ1 was collected by studying datasheets and through development of a demonstrator. For RQ2, qualitative and quantitative data was collected through a questionnaire. Quantitative data originated from ten questions with answers on a Likert scale, as well as three multiple choice questions, whereas qualitative data came from four open-ended questions.

## 3.3 Data analysis

The sampled communication data log was first collected in the ScanaStudio software, as described in Section 3.2. Figure 3.3.1 presents a snippet of that, where a typical command and response sequence is shown. A four byte packet from the master corresponded to two bytes of command code and two bytes of an error detection code called Packet Error Code (PEC), marked as CMD and PEC respectively in Figure 3.3.1.

Since the emulator demonstrator would eventually replace a real system, the behavior of the master-slave communication was important to understand on a higher level than hexadecimal data, i.e. how and why the slave (CSB) responded to the commands from the master (BMU). For this purpose, a Python script was created that could decode the hexadecimal numbers from the CSV file generated by ScanaStudio, and map them to the correct commands and response values. This script decoded each four byte packet from the master, according to the message formats specified in the datasheet of the AFE IC. The script also decoded the different slave responses, as well as recorded time stamps and the time required for transmitting each command.

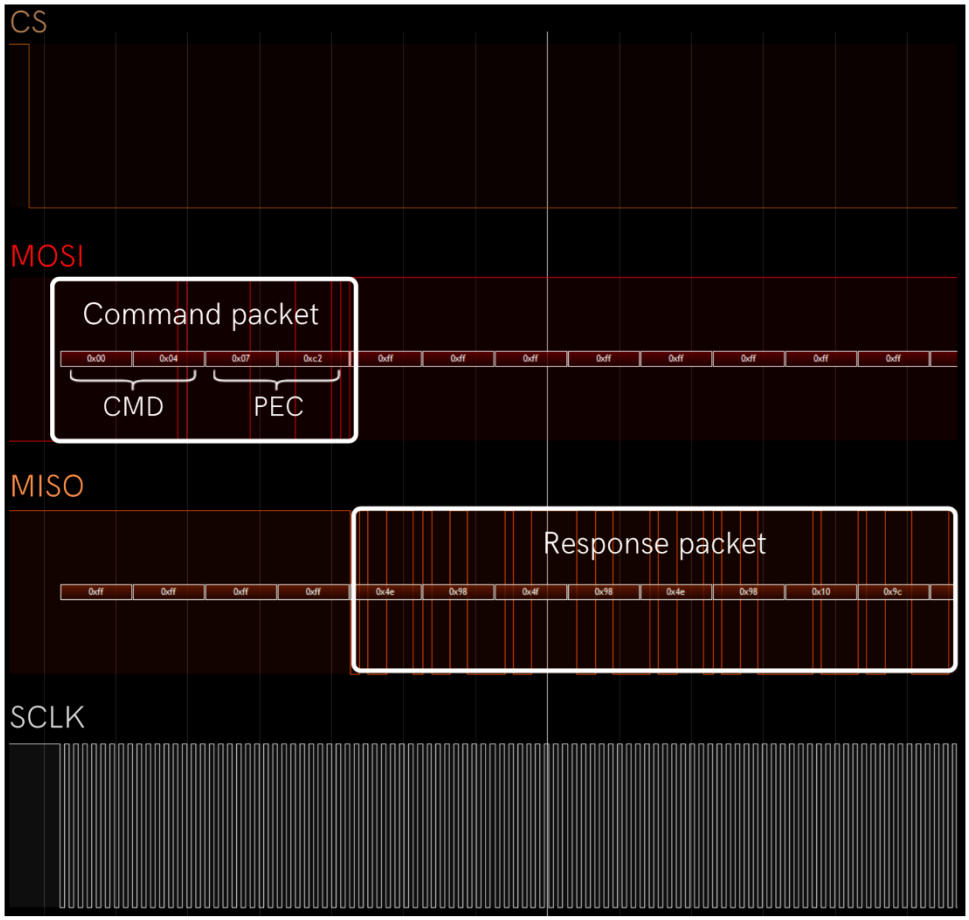



Figure 3.3.1: A typical command and response sequence, as seen in ScanaStudio [30].

The master commands could be divided into poll, read and write types. Poll initiates polling data from the slave, write deals with writing information to registers in the slave's memory, and read regards reading information contained in the slave registers. The different command formats are summarized in Table 3.3.1.

The command-response sequence in Figure 3.3.1 shows how the slave responds to a Read Cell Voltages (RDCV) command. After decoding of the data from the CSV file, the Python script saved all the analysed data in an Excel file.

As seen in the table, the poll command was followed by at least 10 bytes of poll data, but could be more if the slave required more time for polling. In the table, *data bytes* stand for the actual write and read values. For example, in Figure 3.3.1, the slave will respond with measured cell voltages, answering to the read command RDCV. In this case, one slave register stores the data of three cells. The measured cell voltages are stored in 2 bytes per cell, that gives a total of 6 data bytes per slave register as specified in Table 3.3.1. Since the slave does not respond to write commands, it was not in the interest of this study to analyze the content of the data bytes of those commands.

Table 3.3.1: Protocol format descriptions used for decoding of the master-slave communication.

| Command Type | Command Packet | Response Packet |
|---|---|---|
| **Poll** | 2 command bytes, 2 PEC bytes | Poll data (> 10 bytes) |
| **Write** | 2 command bytes, 2 PEC bytes and for each slave register: 6 data bytes, 2 PEC bytes | - |
| **Read** | 2 command bytes, 2 PEC bytes | For each slave register: 6 data bytes, 2 PEC bytes |

#### Research Question 2

The questionnaire respondents were asked to provide their job title, team and years of experience. Their answers were then grouped based on job title, team, years of experience and overall level without regards to those parameters. Their ratings of the importance of high-level and low-level factors underwent statistical analysis in terms of average, as well as non-parametric distribution statistics, to find implications of how the ratings of the factors were correlated to different groups. This was visualized through boxplots, without making any assumptions of the underlying statistical distribution. The questions about emulator usefulness and potential impact on efficiency were analysed on response frequency of each alternative. Lastly, the responses to the open-ended questions were searched for patterns and summarized.

### 3.3.1 Validity

#### Research Question 1

The article *Portal of Research Methods and Methodologies for Research Projects and Degree Projects* states that ”validity, in quantitative research, makes sure that the test instruments actually are measuring what is expected to be measured” [28]. Here, the test instrument for collecting the quantitative data was the logic analyzer, and it was supposed to sample the logic signals at the MCU’s SPI pins to examine the messages sent and received through the isoSPI interface. As a validation method, the logged communication data was decoded. The article also states that ”reliability refers to stability of the measurements and is the consistency of the results for every testing.”

The reliability of the measurements could be guaranteed through a sufficient amount of measurements, since the messages were repeated. Regarding the validity of the case study, delimiting the study to the Northvolt BMS has previously been motivated in Section 1.4. The motivation supports the external validity of the study, in other words the generalizability of the findings, meaning the findings can be applicable for other BMSs using the AFE IC architecture and isoSPI protocol for communication. The demonstrator was validated through test cases, as described in Section 5.2.

### Research Question 2

The respondents of the questionnaire intentionally had different backgrounds, regarding team, job position and years of experience. This would contribute to the external validity, as they represented different perspectives of the BMS testing process. Regarding internal validity of the conclusions made on the responses about the potential impact of an emulator test rig, it could be considered high, since the respondents had at least 1/2 year experience and the majority of their working hours were related to BMS testing. The study was done under highly controlled conditions, that allowed even higher degrees of internal validity. Furthermore, the conducted questionnaire was anonymous which increases the validity of the answers.

# Chapter 4

# Case Study

In this chapter, the case study is presented. It included investigating Northvolt's BMS with focus on the communication between the CSB and the BMU, as well as implementation of a demonstrator emulator. For reasons of intellectual property, the involved hardware and software is not described in detail, but enough information is provided for the reader to follow and review the work.

## 4.1 Northvolt's Battery Management System

The case company has an ambition to be a critical supplier to a sustainable electrification of society through development, production and recycling of LIBs and battery systems. The realization of this ambition sets high requirements on the product in question, as offering a high standard BMS requires a high-quality design on a system level as well as on individual components. The Northvolt BMS was developed through adapting requirements from different LIB applications optimized to perform well in noisy environments, as well as to have high functional safety and accuracy, as previously described in Section 1.4.

Figure 4.1.1 shows the parts of the BMS that have been the focus of this study, divided into two blocks called *BMU master* and *Real System*. The *BMU master* block contains an MCU and an isoSPI interface. The *Real System* block contains several connected CSBs. As shown in the figure, the two blocks communicate through the protocol isoSPI via a twisted pair of wires. The BMU, acting as communication master, dictates the communication.

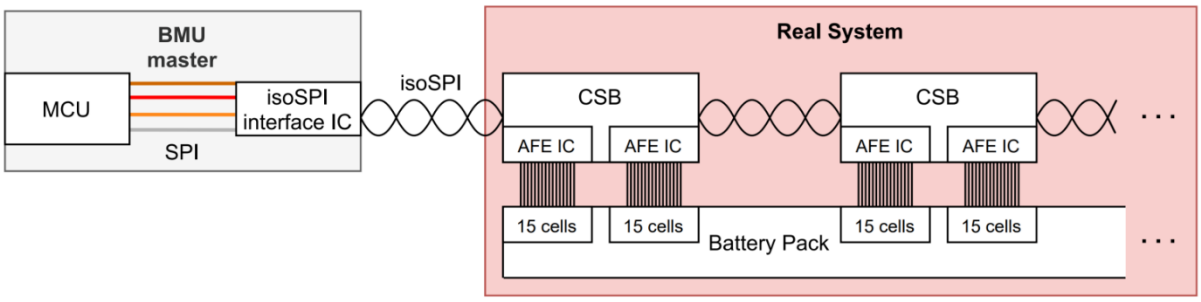



Figure 4.1.1: The part of a BMS in focus of this study, consisting of BMU and CSBs communicating through isoSPI.

The BMU sends its commands in a non-fixed sequence, meaning the order in which the commands are transmitted to the CSBs differ throughout the communication. The CSB must therefore be able to interpret the incoming bits in a way that it can map it to commands and respond accordingly. As opposed to a fixed sequence of commands, the order of the commands are controlled by a state machine and state timers, implemented in the software of the BMU. Furthermore, different functions in the BMS have different priorities. For example, the cell and thermistor voltage measurements have the highest priority and will preempt all other processes. Those are measured frequently within certain time intervals, and diagnostics runs in between. One advantage of having a dynamic BMU of this kind is that it can handle several measurements from theAFE ICs at different time intervals.

A block diagram representation of the AFE IC is presented in Figure 4.1.2, based on the datasheet studied as a part of the case study. As seen in the figure, Multiplexers (MUXs) function as data channels, shared by several cells, transferring the analog signals to the Analog-to-Digital Converters (ADCs). The digital signals then pass digital filters that attenuate high frequencies before they are stored in a memory. The serial communication ports A and B are connected to the memory in order to externally communicate the information stored. Additionally, there are nine General-Purpose Input/Output (GPIO) pins that are used for temperature (also called auxiliary) measurements.

Regarding the memory, the AFE IC has 16 registers containing six bytes of information each, as described in Section 3.3. The registers have specified purposes, including two registers for storing configuration data, five registers for storing cell voltage data, four registers for storing auxiliary voltage data, and two registers for storing status data. The main task of the AFE IC is to provide the BMU with voltage and temperature data,

stored in the registers allocated for that purpose. The reason a BMS requires several AFE ICs is that one AFE IC can only perform measurements on a limited number of cells, in this case 15, whereas a high-voltage battery pack can contain several hundreds of cells.

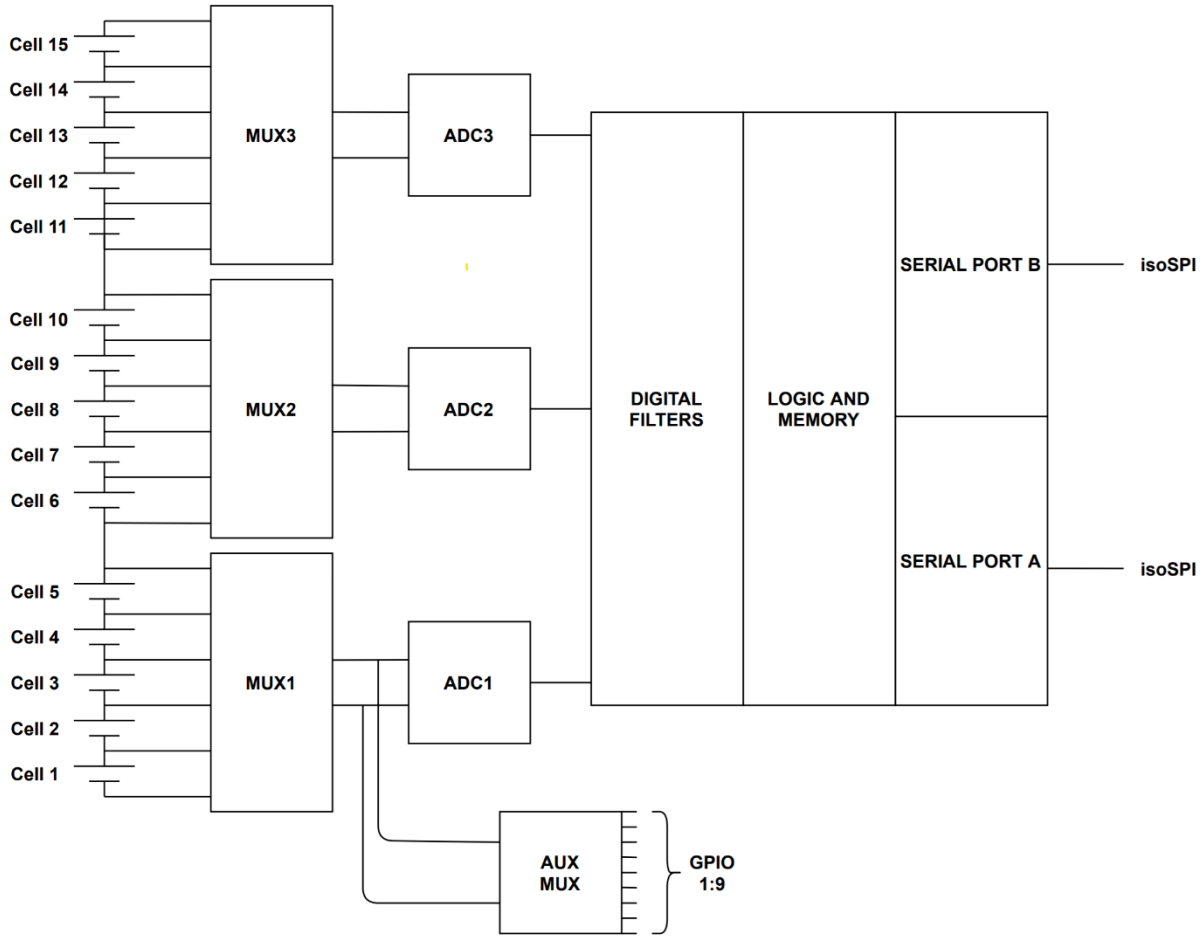



Figure 4.1.2: A block diagram representing the AFE IC of the Northvolt BMS.

### 4.1.1 Decoding communication data

As described in Section 3.2-3.3, the decoding of the sampled data from the CSB-BMU communication was automated through a Python script. The four Command Packet bytes transmitted by the communication master in the MOSI channel translated to a command code and a PEC. The two command code bytes were translated from hexadecimal numbers to commands by studying bus protocols, command formats and command bit descriptions and then mapping them to the right command code. The two PEC bytes were calculated using bus protocols, the PEC format description and then following the PEC calculation description of the datasheet of the AFE IC.

The Response Packet in the MISO channel, containing eight bytes per register for read commands, translated to six bytes of register data and two bytes of PECs. The six bytes of register data was translated by using registers memory maps and memory bit descriptions. The poll command would be followed by at least 10 bytes in the Response Packet, for indication of ongoing analog-to-digital conversion. The slave would respond with 0x00 (logic low) until conversion was completed, then with 0xFF (logic high) to notify the master that the conversion was completed. The master would then stop polling data from the slave.

The Python script would take the CSV file exported from the ScanaStudio software as an input, run the decoding program and then save the translated Command Packets and Response Packets together with the calculated PEC and the time stamps of all sequences in an Excel file, for data analysis purposes as described in Section 3.3.

## 4.2 Emulator Demonstrator

A structured method for the development of the emulator was applied, namely the V-model for software development, illustrated in Figure 4.2.1. It follows a sequential design process similar to the waterfall model. However, in contrast to the waterfall model, the V-model steps are bent upwards midway instead of continuing down in a linear way, thus forming the V-shape. The initial stages of the V-model focuses on the broad design of the software, then progresses through more and more granular stages, leading to implementation and coding, and finally back through all testing stages prior to completion of the project. The reason for applying this systematic approach to the development was to improve the overall quality [31].

Following the first stage of the V-model, the requirements for the migration from real batteries and CSB to emulator needed to be evaluated, a process that overlapped with the stages of data collection for RQ1 described in Section 3.2. Next, it was necessary to find technical ways to create and execute such an emulator, as well as evaluate the emulator performance and impact on the testing process, for RQ2. The first steps leading up to coding are presented here, whereas the steps concerning testing and validation are presented in Section 5.2, Emulator Performance.

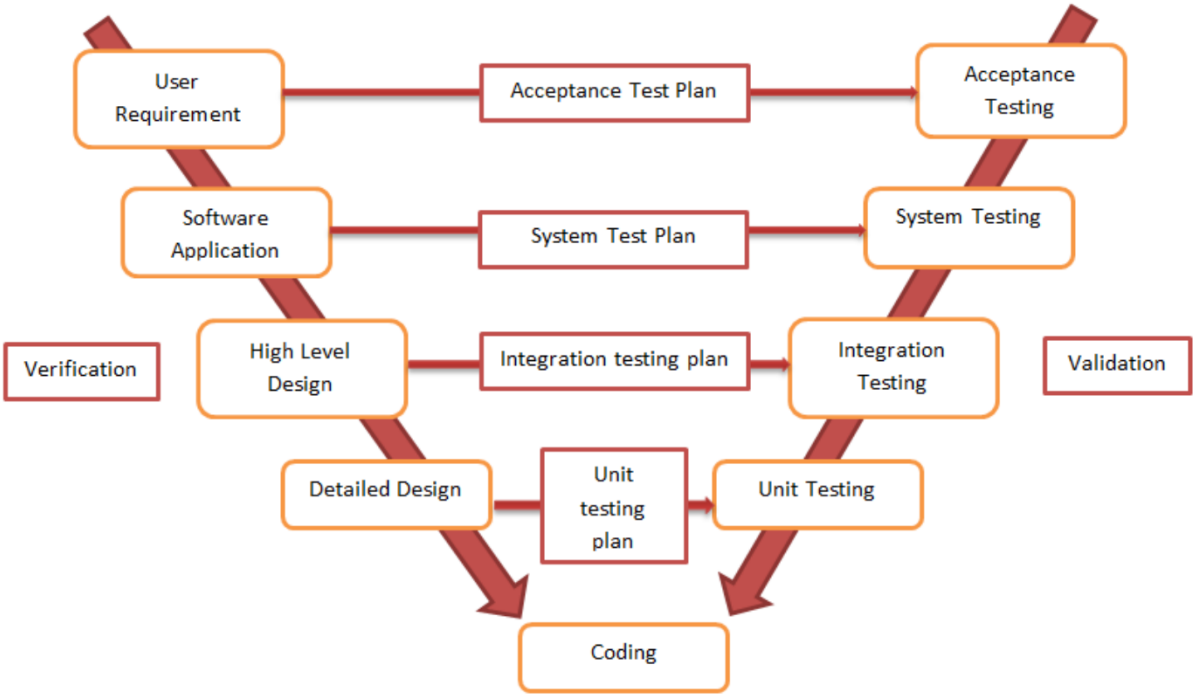



Figure 4.2.1: The V-model for software development in steps [32].

### 4.2.1 Requirements

Mainly three activities of the case study were key in order to form the requirements of the emulator:

1. Study of the datasheet of the IC to be emulated to understand the overall logic and low-level functioning.
2. Study of the embedded code of the BMU, to understand the interaction with the communication master.
3. Discussions with the engineers involved in BMS development at Northvolt in order to delimit and make a priority for the functionalities of the emulator.

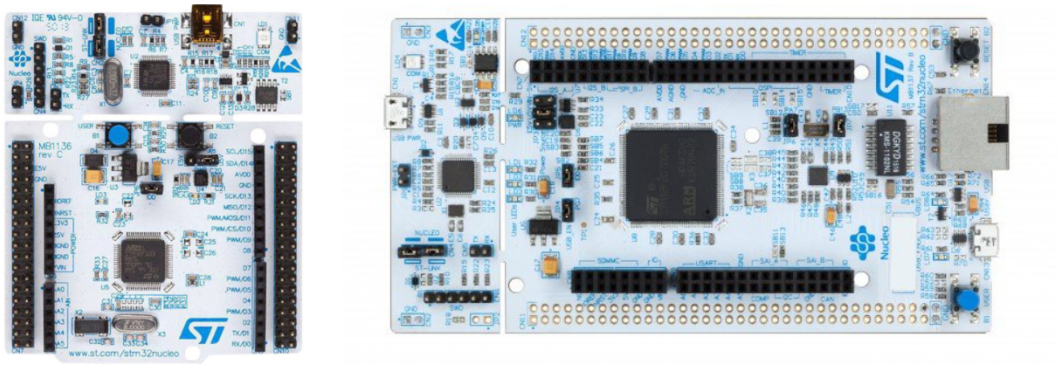
Table 4.2.1 presents the derived requirement specification. The requirements were categorized as functional, non-functional/quality and design constraints. The requirements presented in the table were considered as minimum requirements.

Table 4.2.1: The requirements for the emulator demonstrator.

| Type | Requirement |
|---|---|
| **Functional (FR)** | 1. The emulator must have the properties of a SPI slave, that is: use the functions *receive*, *read* and *reply* when interacting with a master device.<br><br>2. The emulator must reply with adequate data to the following commands from a master device: Poll ADC Converter Status (PLADC), Read Configuration (RDCFG), RDCV, Read Auxiliary Voltages (RDAUX), Read Status (RDSTAT).<br><br>3. The number of emulated AFE ICs and associated cells must be configurable.<br><br>4. The SPI communication should be configured to mode 3.<br><br>5. The emulator must emulate cell data realistically, constant cell voltages and temperatures at a minimum. |
| **Quality (QR)** | 1. The timing of the response must be equal to that of the *Real System*.<br>2. The PEC must be calculated according to the algorithm defined in the emulated IC datasheet. |
| **Design (DR)** | 1. The hardware used for development must be NUCLEO-F103RB as master device and NUCLEO-F746ZG as slave device (emulator), each connected to isoSPI interfaces.<br><br>2. The physical communication must be through isoSPI. |

### 4.2.2 Development setup

According to Design Requirement (DR) 1, the hardware setup used for development was divided into two parts, one MCU for running the emulator code and one MCU acting as BMU and communication master, hereafter called CSB master. The emulator MCU was the development board NUCLEO-F746ZG, and the CSB master MCU was the development board NUCLEO-F103RB. The boards can be seen in Figure 4.2.2. The Nucleo boards are affordable and flexible MCUs for building prototypes with a large variety of features [33, 34].



(a) The CSB master. (b) The emulator.

Figure 4.2.2: The development boards for the emulator (NUCLEO-F746ZG) to the right and CSB master (NUCLEO-F103RB) to the left [33, 34].

Figure 4.2.3 illustrates the development setup containing the two development boards denoted as MCUs. Both development boards were connected to isoSPI interfaces as required.

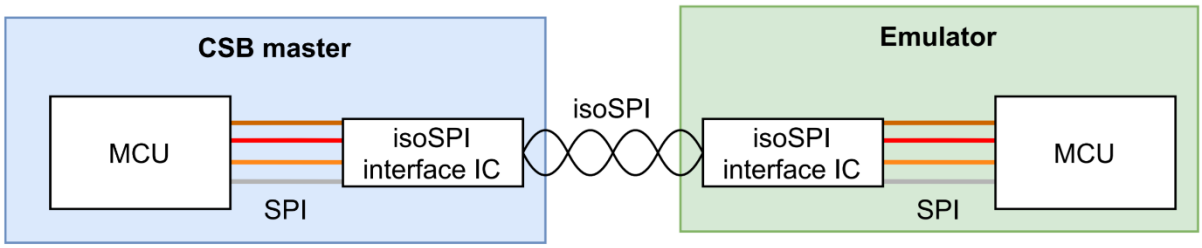



Figure 4.2.3: A schematic representation of the setup used for the developing of the emulator

The emulator code was written in C++ in the mbed Studio environment. The reason for selecting DR1 and the coding environment was the compatibility with the company BMS.

### 4.2.3 Detailed design and implementation

The design and implementation phases of the demonstrator development went through two main iterations before the emulator worked in a satisfying way.

#### First iteration

The code structure presented as a flowchart in Figure 4.2.4 represents the first version of the demonstrator. The slave (emulator) would be polling bytes from the master with the `spi.receive()` function, continuously ”listening” for commands to be received at its SPI pins. The bytes received would be saved in an array called `cmd` with the function `spi.read()`. The slave would read 4 bytes at a time to be able to interpret commands. When the array `cmd` would be filled with 4 bytes, the slave would perform error detection by checking the PEC value that is calculated from the last two bytes in the command. If the PEC was valid, the message was correct. The first two bytes in the Command Packet would then be mapped to a command in the switch-case. The slave would then answer with an array of bytes, called `response` in the flowchart, with different content depending on the case, through the function `spi.reply()`. The program would then loop back and continue to poll commands from the master.

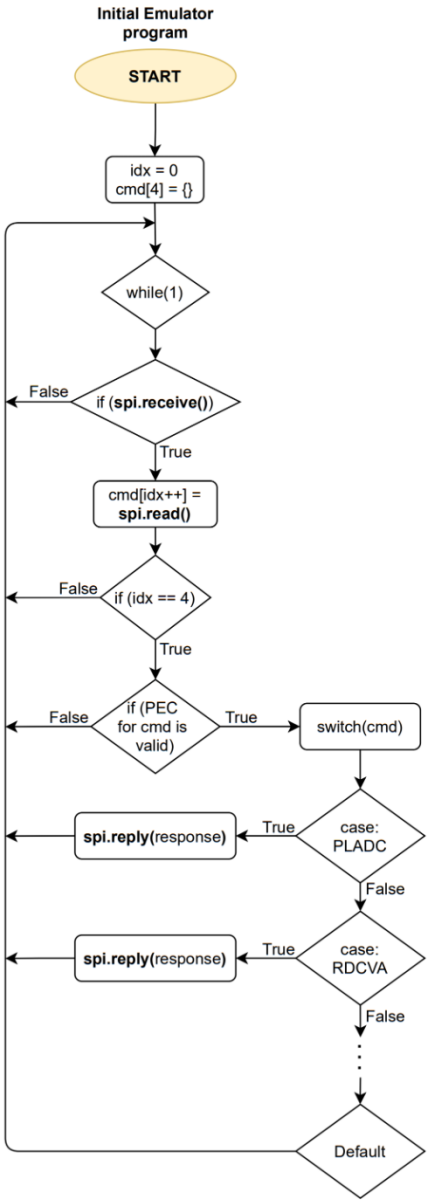
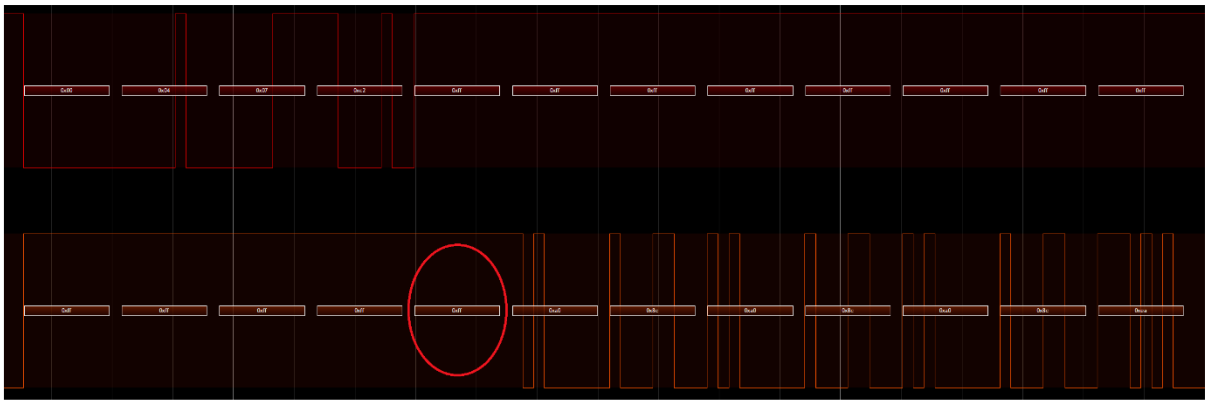



Figure 4.2.4: A flowchart of the main program of the emulator code.

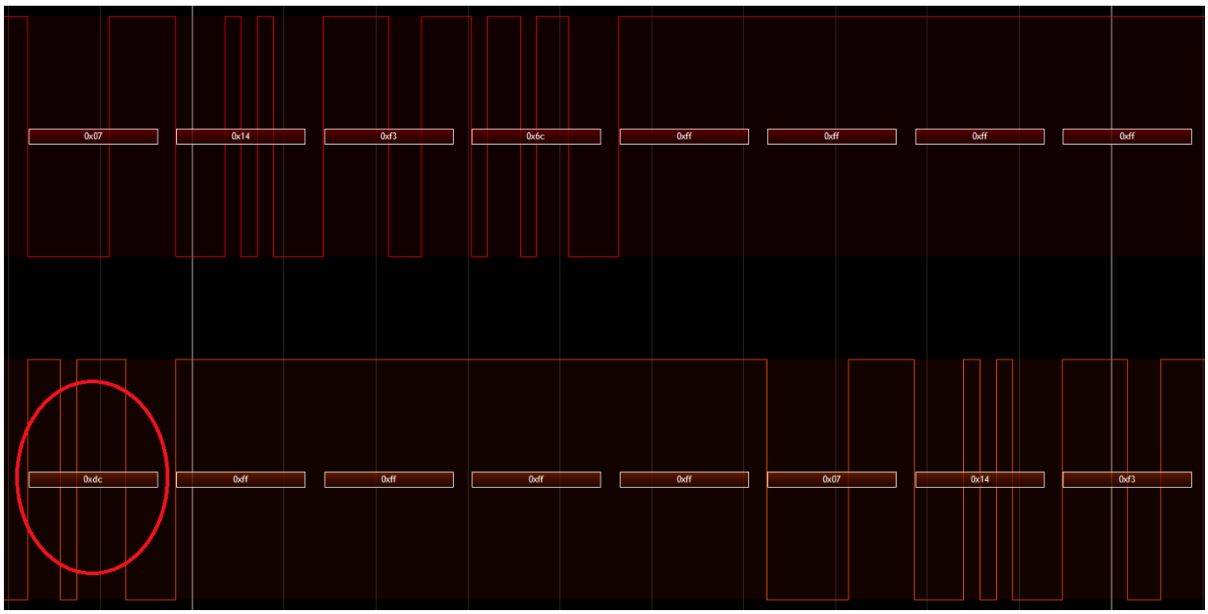
However, this approach did not fulfill the basic requirements for the isoSPI communication. The reason being that the slave was not able to transmit the responses

fast enough, resulting in bytes that got shifted during the communication. This version of the emulator worked successfully for around 500 ms of communication, which corresponded to approximately 500 command and response sequences, before the faulty behaviour was observed.

The main issue was that an extra byte was added to the slave reply buffer, shifting one byte for all subsequent bytes to be sent. The next byte that the slave sent would be from the previous command. The shift happened when either one 0xFF was added while polling, or one byte was repeated in the reply, as seen in Figure 4.2.5 and 4.2.6 respectively.

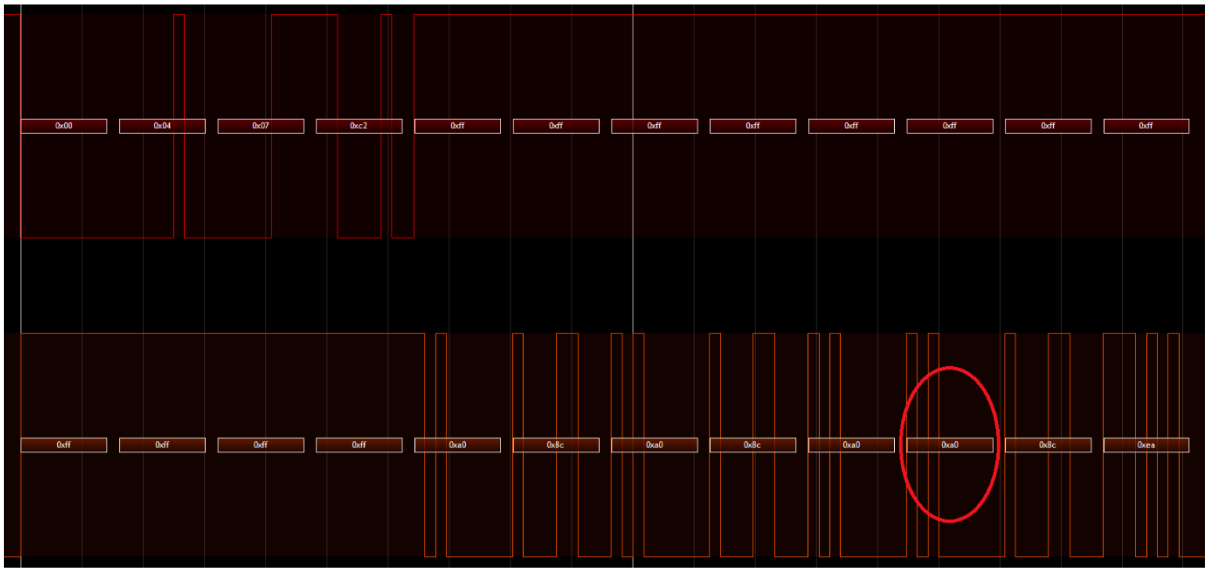


(a) An extra byte, 0xFF, was sent from the slave.

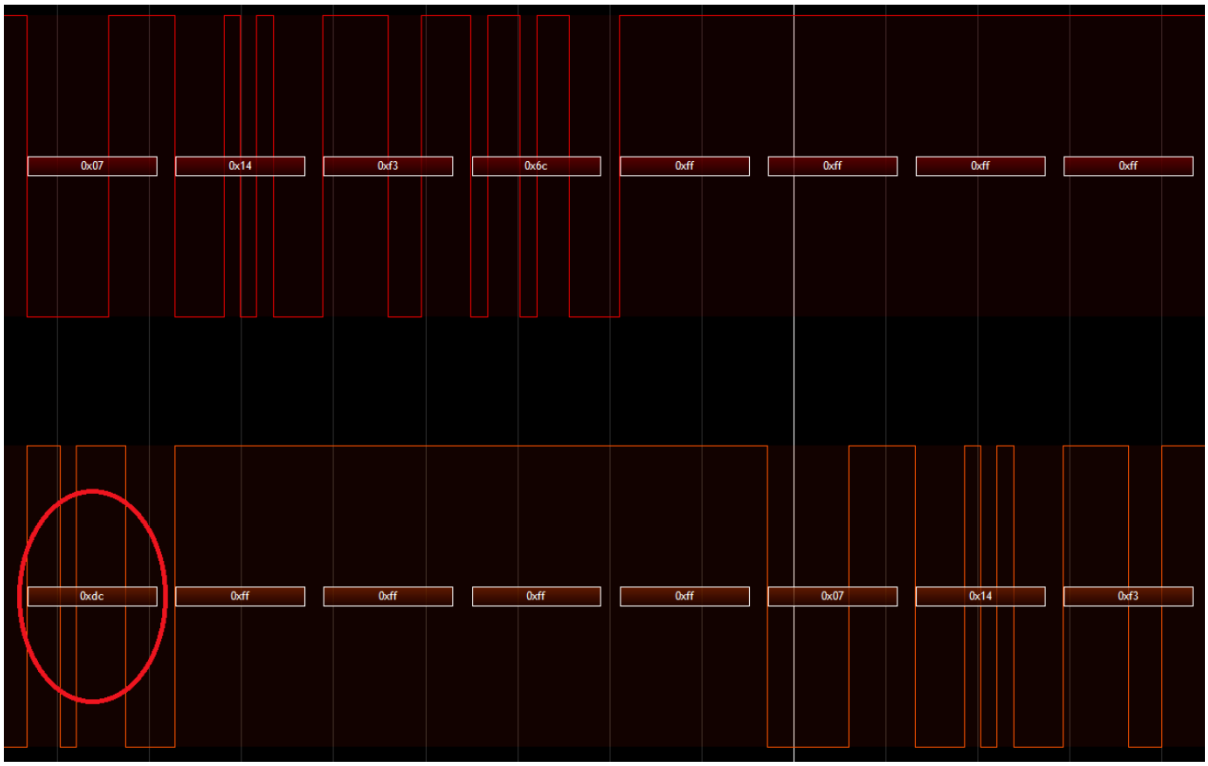



(b) The following bytes are shifted by one.

Figure 4.2.5: The effect of one extra 0xFF byte in the slave response, captured by ScanaStudio [30].




(a) An extra byte, 0xA0, was sent from the slave.




(b) The following bytes are shifted by one.

Figure 4.2.6: The effect of one extra 0xA0 byte in the slave response, captured by ScanaStudio [30].

The issue with excessive bytes that interrupted the synchronization between master and slave was attempted to be fixed by e.g. decreasing the SPI communication frequency, adding delays, skipping calculations and preparing the bytes before initializing SPI communication to decrease software delays. Nevertheless, the issue kept appearing and led to a second iteration, where a new approach was applied.

### Second iteration

Due to the limitations found during the first iteration, some of the initially stated requirements were reconsidered and it led to some changes in the functionality and implementation of the emulator. The most significant differences between the two iterations are listed below.

- The low-level Hardware Abstraction Layer (HAL) library with enabled Direct Memory Access (DMA) for SPI communication was used instead of the previously used functions from the mbed SPI framework.
- The emulator responses would be sent in a *fixed sequence* stored in a buffer before transmission, meaning the emulator would require the commands to be sent in a corresponding fixed sequence from the master as opposed to being continuously updated. This was not the case in the current Northvolt BMS, as described in Section 4.1.
- There would be no error detection performed by the emulator, meaning that the slave would provide the master with an answer regardless of the PEC being correct or not.
- Two interrupt service routines were added.

The second version emulator contained two DMA buffers, called transmit and receive, with sizes twice as long as the longest slave response, meaning that there would always be room in the buffers to store at least two responses. Response bytes filled the whole DMA buffer from the beginning. When SPI communication had been initiated by the master, the slave began a continuous transfer, where the transmit DMA sent data to the SPI transmit buffer and the receive DMA buffer received data from the SPI receive buffer simultaneously on each SPI clock signal. Then both DMA-buffers incremented with index 1, until they reached the end, where it pointed back to the beginning.

Additionally, two interrupt service routines were included, to ensure that the DMA buffers were always ready with the next response. The interrupt function `HAL_SPI_HalfDone` was triggered when the first half of the DMA buffers had been traversed, so the next data could be filled in both DMA buffers. The interrupt function `HAL_SPI_Done` was triggered when the whole DMA buffer was filled, that is when it would increment back to index `0`. Thus making the communication rely solely on the hardware, as the software made sure to always keep the DMA buffers filled. Both

interrupts were therefore triggered by the hardware, preempting the main program and initiating new preparations of the outgoing buffer in advance, to achieve the desired timings.

The second iteration code structure is presented as a flowchart in Figure 4.2.7. A response list that contains all responses would be prepared first, which would emulate the structure of the AFE ICs memory registers. Responses were sent in the same order in the `response_list`. Thereafter, SPI communication would be initialized and configured on the HAL level. The variable `currCmdIdx` starts from the last index in `response_list`, so the next index would be the first in the list, thus making `response_list` work as a *circular buffer*. The `buffer`, which contained the response bytes, were sent with the corresponding `bufferSize` to the function `PrepareOutgoingBuffer`. After the buffer preparation, the HAL SPI transmission and reception would take place.

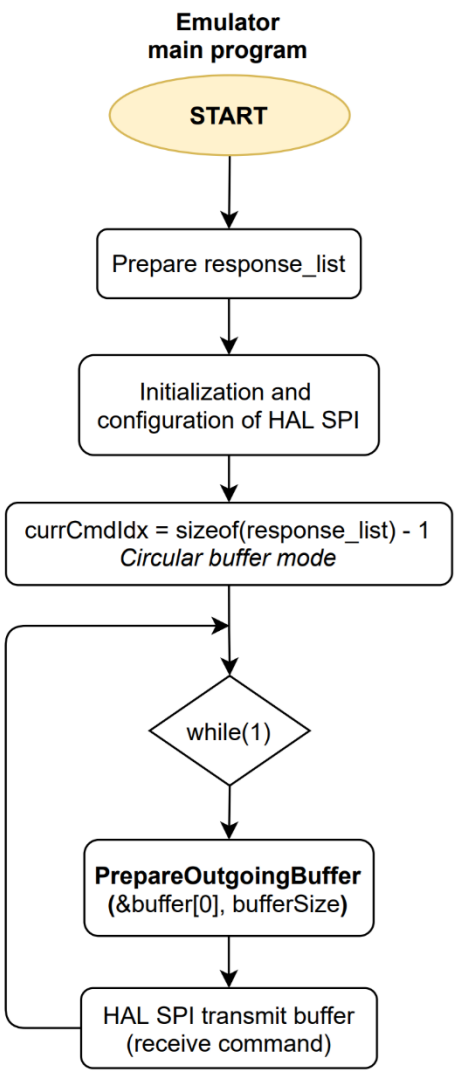



Figure 4.2.7: A flowchart of the main program of the emulator code.

The flowchart of the `PrepareOutgoingBuffer` function is presented in Appendix A. The `currCmdIdx` would iterate through the whole response list, going to the next index in the `response_list` each time the function `PrepareOutgoingBuffer` is called. The flowcharts of the two interrupts are shown in Figure 4.2.8. All of the variable names contained in the flowchart are described in Table 4.2.2.

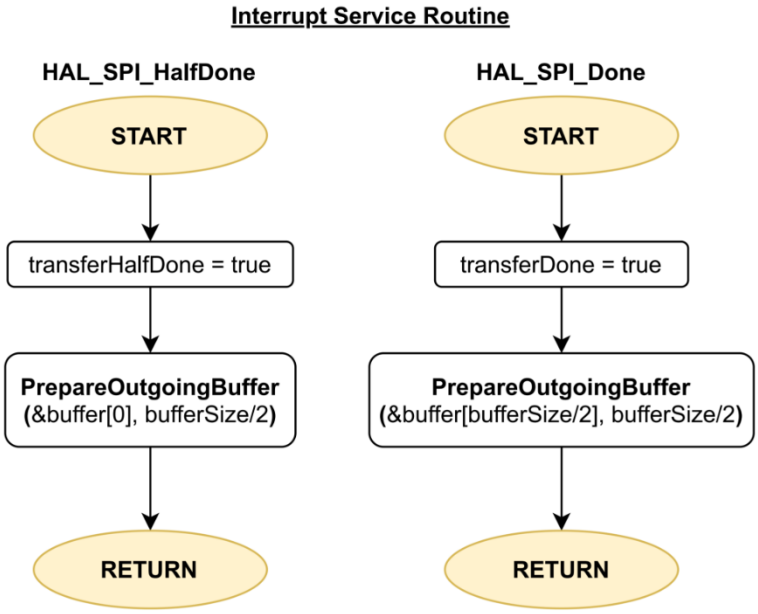



Figure 4.2.8: Flowcharts of the interrupts of the emulator code.

Table 4.2.2: Descriptions of variables from the emulator code flowchart.

| Variable | Description |
|---|---|
| `buffer` | Contains the response bytes |
| `bufferSize` | Size of buffer |
| `bytesLeftFromPrevious` | The bytes that didn't fit in the other half of the buffer |
| `bytesTotal` | The amount of bytes transmitted at a certain time |
| `currCmdIdx` | The current index of the reponse list |
| `incomplete` | Flag for stating if bufferSize is updated |
| `response` | Response to a certain command |
| `response_list` | List of responses |
| `responseSize` | Size of response_list |
| `size` | Number of bytes to be added to bufferSize |
| `transferDone` | Flag for stating if tranfer is done |
| `transferHalfDone` | Flag for stating if (half) tranfer is done |

The demonstrator was connected to an emulated master throughout development, running on a NUCLEO-F103RB as stated in DR1. The code for the master emulator was developed in parallel with the emulator. The flowchart of the emulated master code is shown in Figure 4.2.9, and the variables are described in Table 4.2.3.

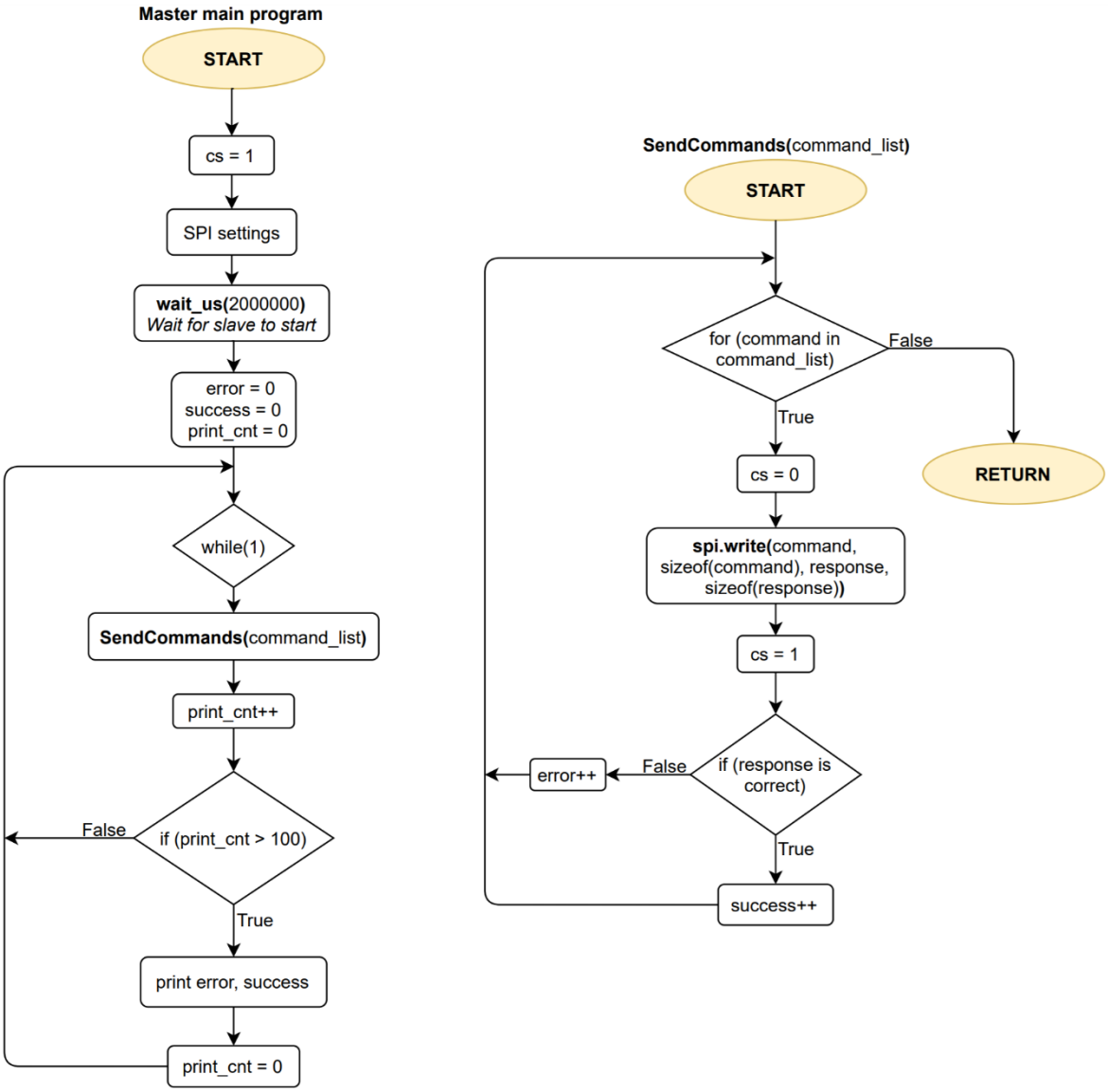



Figure 4.2.9: A flowchart of the emulated master code.

The master program sent commands in the order of the command list, that corresponds to the response list in the emulator program. All commands in `command_list` were sent in the function `SendCommands` with `spi.write()`. After sending the command and receiving the response, CS was turned to logic high, equal to binary `1`, to end the communication. The master would then check if the response that was received was correct, increasing 1 to the value of the variable `success`, or, if the response was incorrect, adding 1 to the value of the variable `error`. These values were then printed in the main program for the user to verify successful communication.

Table 4.2.3: Descriptions of variables from the master code flowchart.

| Variable | Description |
|---|---|
| `command` | Command to be sent to emulator |
| `command_list` | List of all commands |
| `cs` | Chip select, dictates the isoSPI communication |
| `error` | Counter of faulty responses |
| `print_cnt` | Counter to print the error rate |
| `response` | Actual response from emulator |
| `success` | Counter of successful responses |

# Chapter 5

# Results

The results are presented in this chapter. This includes findings from the case study, including demonstrator integration, test and verification, as well as an analysis of the questionnaire.

## 5.1 Northvolt's Battery Management System

In relation to the theoretical background presented in Section 2.1, the Northvolt BMS is characterized by the following.

- **Functionality:** The Northvolt BMS covers all of the functionality listed in Section 2.1.1. The AFE IC specifically is in charge of gathering cell voltage data and temperature data. It also provides the BMU with additional data that is not relevant to the overall BMS functionality but rather the implementation, such as IC self diagnostics and configuration data through the RDSTAT and RDCFG commands, as described in Section 4.
- **Architecture:** The Northvolt BMS has a centralized architecture. As stated in Section 2.1.2, it means that there is one central control unit, referred to as BMU throughout this thesis. The communication between CSB and BMU is implemented in a master-slave configuration of a daisy-chain topology.
- **Communication:** The link between CSB and BMU is isoSPI as it has advantages over other options, explained in Section 2.1.3. The implementation of isoSPI was a pre-requisute for choosing the Northvolt BMS as case company, as stated in Section 1.4.

## 5.2 Emulator Performance

In order to compare the emulator with the real system, the emulator performance was evaluated with regards to fulfillment of the requirements. Table 5.2.1 presents how the Functional Requirements (FRs), Quality Requirements (QRs) and Design Requirements (DRs) were tested, along with the result for each test.

Table 5.2.1: Tests for verifying the fulfillment of demonstator requirements.

| | Test | Result |
|---|---|---|
| **FR1** | Analyse the master-slave interaction | Correct behavior, verified with ScanaStudio |
| **FR2** | Compare a log from the emulator with the BMU log | Responds as expected, verified with ScanaStudio |
| **FR3** | Count the number of emulated cell voltage readings through SPI decoder in ScanaStudio | Number of AFE ICs set in code corresponds to the same number of AFE ICs in the sampled data, verified in ScanaStudio |
| **FR4** | Configuration + logic analyzer sample showing phase and polarity | Phase and polarity corresponds to mode 3, verified in ScanaStudio |
| **FR5** | Compare a log from the emulator with the BMU log | Constant cell measurements, verified with ScanaStudio |
| **QR1** | Use markers in ScanaStudio to measure the timings | Timing results are presented in Table 5.2.2 |
| **QR2** | Compare emulated PEC with real PEC from BMU data, through print in serial monitor | The calculated PEC is equal to PEC in the BMU, verified through prints |
| **DR1** | Schematic figure of the setup | Figure 4.2.3 |
| **DR2** | Schematic figure of the setup | Figure 4.2.3 |

It should be noted that the emulator was connected to the CSB master used for emulator development when testing, instead of the real BMU as was the initial ambition. Thus, the valid schematic figure for the verification was the same as during the development, presented in Figure 4.2.3. This meant that regarding system integration with the real BMU, this was not accomplished due to the fact that the emulator expected a fixed sequence of commands, as explained in Section 4.2.3.

The FRs were all accomplished, as verified by observing the communication signals sampled by a logic analyzer in ScanaStudio. The signals were sampled using a similar method as the one described in Section 3.2, but connecting the logic analyzer to both the master and the slave end of the communication line. The PEC calculation could easily be tested as a separate module and simply be verified by printing the calculation result in the serial monitor, which resulted in a successful QR2 test. The two DRs were verified with figures, as stated in the table. All of Section 4.2.2 supports the verification of the two DRs.

Table 5.2.2: Comparing the timings of six different measures between the *Real System* and the emulated system.

| **Timing measure** | **Real system** | **Emulated system** |
|---|---|---|
| **SCLK frequency** | 800 $kHz$ | 1130 $kHz$ |
| **Master high, slave first bit** | 40 $ns$ | 15 $ns$ |
| **CS low, MOSI low** | 4.19 $\mu s$ | 8.9 $ns$ |
| **CS high, MISO high** | 95 $ns$ | 110 $ns$ |
| **1 byte transmit** | 10.1 $\mu s$ | 6.67 $\mu s$ |
| **Between two bytes** | 1.30 $\mu s$ | 2.39 $\mu s$ |

In order to verify QR1, that is, if the timings of the emulator response were equal to that of the real system, different timing measures were defined. These are listed in Table 5.2.2. As seen in the table, none of the timings are equal, thus the requirement was not fulfilled. However, one can argue that the differences in timings listed in the table can not be fully compared because the communication measured is driven by the communication master, which in the emulator case is the CSB master instead of the BMU. Since the hardware, software and SCLK frequencies differed between the BMU and the CSB master, differences in the timings were expected. Except for the time between CS high and MISO low, and between two bytes, the timings of the emulator were faster than that of the real system, which implies that there is room for fulfilling the requirement if integrating the emulator with a real BMU, contrary to if the two were reversed. The results are further discussed in Chapter 6.

## 5.3 Hardware limitations

Initially, the origin of the timing issues that were manifested with the first iteration of the emulator were unknown. After the implementation of the second iteration, informally conducted research outside of the scope of the V-model development, confirmed that the emulator hardware was the root cause of the timing issues. One problem was that the slave development board did not manage to read in bytes from the master and also prepare the right answer sequence in time. In addition to that, it had a transmit First In, First Out (FIFO) buffer that was automatically filled with bytes as the prepared answers were delayed, disrupting the buffering controlled by the emulator software. The first iteration code structure showed promising results on another development board that did not have a buffer. However, the second iteration code had already been tested and verified at this stage of the project, leaving no room for further fine-tuning. Recommendations for continued development of the emulator are elaborated in Section 6.5, Recommendations to the Company.

## 5.4 Questionnaire results

96.2% of the questionnaire target group responded, resulting in a diverse and balanced representation of different backgrounds: software, application, integration, Quality Assurance (QA), system and hardware. For analysing the answers, they were grouped on three factors: job title, team and years of experience. Table 5.4.1 presents the groupings, as well as the ratios between different groups. The factors rated in the questionnaire are presented in Table 5.4.2, with high-level factor (HLF) being general features and low-level factor (LLF) being implementation specific features. These factors were derived in the same way as the requirements for the emulator, see Section 4.2.1. The respondents ratings of the importance of the factors are presented as box plots in Figure C.0.1 and C.0.2 in Appendix C. The average values of the ratings as well as the most outlying average values of the group's ratings of High-level factors (HLFs) and Low-level factors (LLFs) are summarized in Table 5.4.3 and 5.4.4. As the two tables show, software team/job title have the opposite extreme average value than other team/job title in majority of the cases. This goes for HLF 1-3 and LLF 1 and 3-4, or 60% of the cases, where the group Other have rated almost all factors above the overall average and the software group have rated factors below the overall average.

Table 5.4.1: The categorized respondents and the ratios between groups.

| Category | Grouping | Ratio |
|---|---|---|
| **Job title** | Software/Other | 62%/38% |
| **Team** | Software/Other | 50%/50% |
| **Years of experience** | <2/>2 | 54%/46% |

Table 5.4.2: The factors rated with regards to importance in an emulator test rig.

| Factor | Description |
|---|---|
| **HLF1** | Cell voltages and temperatures can be manually set to values of your choice |
| **HLF2** | Cell voltage and temperature measurements are not constant values, but have a realistic behaviour (drive cycle) |
| **HLF3** | Flexibility: The number of CSBs can be configured as you want |
| **HLF4** | User-friendliness: Easy to setup and run the emulator (<15 min to set up) |
| **LLF1** | The emulator verifies the received command by checking the PEC |
| **LLF2** | The emulator replies to Read Status Register command |
| **LLF3** | It is possible to clear the Cell/Aux/Status Register groups |
| **LLF4** | It is possible to write values to the emulated memory (Write Configuration command) |
| **LLF5** | The emulator replies to Read Configuration command |
| **LLF6** | The emulator replies to Poll ADC Conversion Status |

Table 5.4.3: The average rating of each high-level factor with regards to all respondents, and the most outlying averages on group level.

| HLF | Average | Max (group) | Min (group) |
|---|---|---|---|
| **1** | 4.38 | 4.47 (Software job title) | 4.22 (Other job title) |
| **2** | 3.58 | 4.56 (Other job title) | 3.00 (Software job title & team) |
| **3** | 3.83 | 4.11 (Other job title) | 3.67 (Software job title) |
| **4** | 3.71 | 4.33 (Other job title) | 3.31 (<2 years of experience) |

The respondents also had the opportunity to add factors/features that they considered important for an emulator test rig through an open-ended question (see Appendix B). The responses are summarized in the list below.

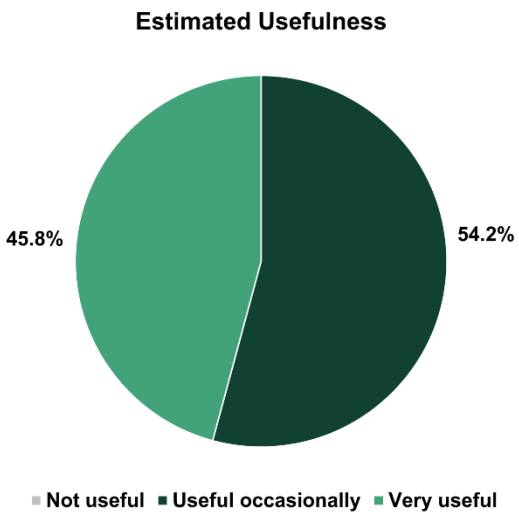
Table 5.4.4: The average rating of each low-level factor with regards to all respondents, and the most outlying averages on group level.

| LLF | Average | Max (group) | Min (group) |
|---|---|---|---|
| **1** | 3.12 | 3.67 (Other job title) | 2.50 (Software job title) |
| **2** | 3.76 | 4.14 (>2 years of experience) | 3.50 (<2 years of experience) |
| **3** | 3.38 | 3.67 (Other team) | 3.00 (Software team) |
| **4** | 3.75 | 4.00 (Other team & >2 years of experience) | 3.43 (Software team) |
| **5** | 3.71 | 4.00 (>2 years of experience) | 3.43 (Software team) |
| **6** | 3.53 | 3.71 (>2 years of experience) | 3.40 (<2 years of experience) |

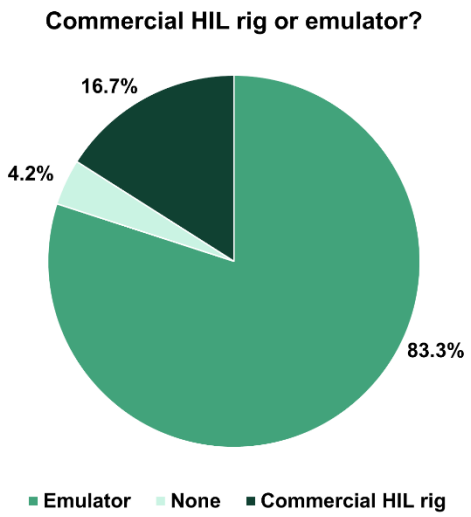
1. Emulate cell balancing bits (one respondent)
2. Network infrastructure/internet connectivity (one respondent)
3. Writing to status registers possible (one respondent)
4. Emulate IC diagnostics (two respondents)
5. Auto reset (one respondent)

Apart from rating and suggesting factors for an emulator test rig, the respondents were asked to estimate: how useful it would be for them; how much of their working hours could be used more efficiently if the emulator was available; and if they would choose to have an emulator or a commercial HIL rig at hand at work (or none of these). The responses are presented as charts in Figure 5.4.1. Below is a summary of the responses to these three questions.

- **Usefulness:** The respondents had three choices, but since no one answered that an emulator would not have any use at all for them, only two alternatives are viewed in the chart. The answers were almost equally divided between ”Very useful” and ”Could be useful occasionally”.
- **Efficiency:** The respondents estimations of how much of their working hours could be used more efficiently (in percent) with an emulator test rig at hand resulted in 88% of the respondents answering 25% or more.
- **Choice:** When choosing between a commercial HIL rig and an emulator test rig, a notable majority would have chosen the emulator.




(a) The estimated usefulness.




(b) Choice: HIL rig or emulator.

Figure 5.4.1: Respondents answers to questions about usefulness and which test rig is preferred.

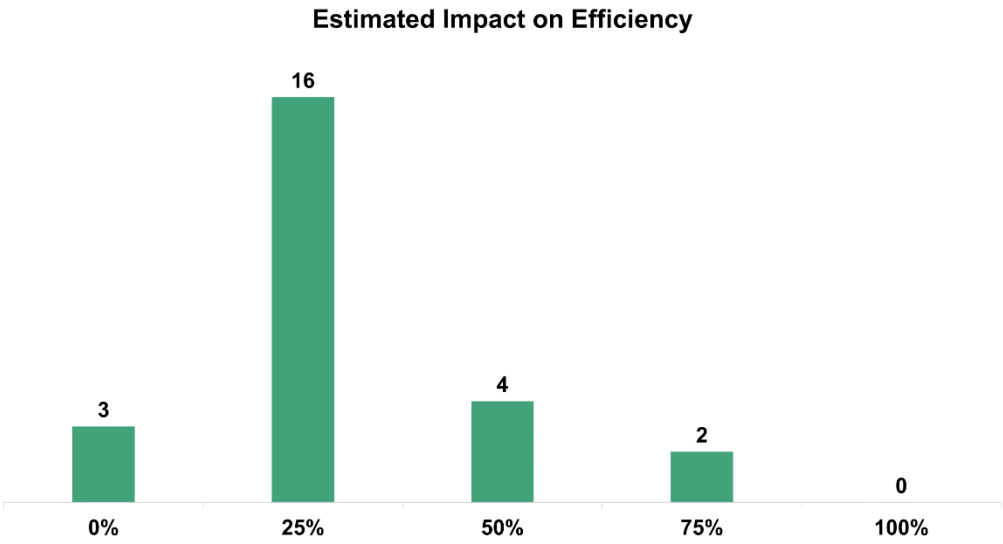



Figure 5.4.2: The estimated impact on efficiency.

The two open-ended questions of the questionnaire were:

1. How, if at all, would having such an emulator save you time in your work with BMS testing?
2. Could you give at least one example of a case where the emulator would be too limited for BMS testing?

The answers were unified and are summarized below.

1. Many respondents expressed two main issues with the current setup, that it is not accessible enough and that it is not flexible enough. The emulator would solve the first issue by enabling, as the respondents state, testing from home or on the bench; testing in parallel with other engineers; testing small software changes; debugging in an early stage; informal testing. The second issue would be solved by the emulator by enabling testing for an arbitrary number of cells; testing scenarios that are not reachable with real cells; testing with noise-free digital signals.
2. The respondents came up with multiple cases for which the emulator would be too limited, e.g. hardware failure states; parameter estimation for SOC and SOH algorithms; specific faults under specific conditions; final verification; testing for CE marking; diagnostics testing. Only one respondent stated that the level of complexity of an emulator would be enough for them in their role.

Finally, the respondents were asked to share what difference they thought a full CSB + battery emulation would make in their work, compared to a limited emulator (fulfilling the requirements stated in 4.2.1). 36% stated that it wouldn't make a big difference, or with the words of one respondents: "the simple emulator is still a big step forward". 12% answered not sure or nothing. The rest of the respondents gave answers reflecting the answers of the second open questions about cases where the emulator would be too limited.

# Chapter 6

# Discussion and Conclusions

In this chapter, the results from the previous chapter are evaluated for answering the RQs. The positive effects, drawbacks and overall impact of the project are discussed, as well as suggestions for future work.

## 6.1 Research Question 1

*What are the factors to consider when creating an Analog Front-End (AFE) isoSPI communication emulator for a highly accurate BMS like Northvolt's BMS?*

The results from the case study showed that the specific functionality, architecture and communication of the system to be emulated and integrated with the emulator all entail important factors for consideration when creating an Analog Front-End communication emulator for a highly accurate BMS like Northvolt's BMS.

### 6.1.1 Functionality - static cell data vs. dynamic cell model

From the questionnaire answers about the ratings of the importance of different features of an emulator, it seemed that the rating in some cases depended on the respondents engineering discipline, but in some cases no such conclusion could be made since the spread within a group was large. For the HLFs, the average rating was never below 3, with the exception of HLF4 (user-friendliness) which the group Software job title rated below 3 with regards to average. The fact that all of the factors were rated medium important (3) or higher, measuring by average, could be a sign that

all of the features are important generally speaking, without regards to any specific BMS testing scenario, and should be considered when creating an emulator.

The most interesting observation regarding the HLFs was the comparison between the rated importance of an emulator providing static cell data (HLF1), versus a dynamic cell model data (HLF2). All respondents agreed that HLF1 was important, a conclusion made since half of the respondents rated 4 or 5, and no one answered below 3. This was not surprising since the capability of providing cell data can be considered as fundamental for an AFE IC emulator. Regarding HLF2, the spread was large, both between groups and within groups. One possible conclusion that can be drawn is that the purpose of the emulator should be clearer in earlier stages of development: who is it developed for and what type of testing will it be used for. Evidently, the importance varies between individual cases. The software engineers had very different opinions on whether or not it was important for an emulator to provide a dynamic cell model, resulted in an average rating 3. Comparing that to the answers of the group *other teams*, it is clear that on a group level they had a more unified view, rating the dynamic cell model higher with a median above 4.

Regarding the rating of LLFs, the results are more uncertain, since it was an optional question that had fewer respondents, especially when divided into subgroups. In addition to that, from an informal conversation with an engineer, it was stated that no more than four of the respondents were familiar with the LLFs. The fact that 18 respondents rated the LLFs further makes the results uncertain, as it is unclear how knowledgeable the respondents were when rating. No conclusions were therefore drawn on group level.

The LLFs are closely tied to the case BMS, so concluding the importance of each of them on a general level requires investigations of other BMSs. What can be stated about the LLFs, however, is that they all had a median rating of between 3 and 4, implying that they are important factors for consideration when creating an emulator in this case. LLF1 (emulator verifies the command by checking the PEC) stood out as the ratings were significantly more spread compared to other LLFs. The different opinions could be related to the respondents varying familiarity with PEC. One explanation for the low ratings could be that some engineers limited contact with PEC in their work, or that they strictly viewed the emulator test rig as an alternative to the HIL rig, which is currently not used for PEC verification.

### 6.1.2 Communication - timing

From the implementation and results, it was evident that timing of communication signals transmitted by the emulator is a highly critical factor when developing a communication emulator. When timings were off, the communication was not synchronised. That resulted in behaviours that were undesirable, making the emulator not fulfilling any of the FRs. One important aspect of this factor is the robustness and repeatability of the timing. It was not stated explicitly as a requirement that the emulator must be synced with a master for longer than 500ms, but rather generally accepted that a useful emulator has a repeatable and robust behaviour towards the interfacing system.

As described in 4.2.3, when applying high-level functions from the mbed SPI library, the desired performance was not achieved because of repeated timing issues that could not be solved through simple software interventions. Since the hardware was already set as a part of the requirement specification, further software modifications were evaluated, resulting in the second iteration emulator. This new approach with HAL and enabled DMA solved the timing issue that was previously observed, though with some compromises. The emulator demonstrator with compromises was acceptable for achieving proof-of-concept and answering of the RQs, but could not be integrated to any existing system and can be considered a drawback of the study.

The timing measures used for verifying QR1, presented in Table 5.2.2, can be considered redundant, as it is ultimately the communication frequency, SCLK, of the *Real System* and the emulator that have to be equal in order for all the other measures to be equal. To make sure that the processor running the emulator is at least as fast as the SCLK on the BMU can be considered an important factor for consideration. Nevertheless, compatibility between the slave's and the master's MCU can be considered a fundamental pre-requisite for development of an emulator, rather than a factor for consideration.

### 6.1.3 Development hardware and software

The hardware used for development of an emulator proved to be an important factor for consideration. It can be considered a mistake to set the development hardware as a design requirement as opposed to make it a design choice after setting the QRs. Using other hardware would most likely not result in the same issues observed in this project,

though it could present other problems instead. From the findings described in 5.3, it can be concluded that how the SPI communication is implemented in the development board should be reviewed in an earlier stage, specifically whether or not a buffer that is automatically filled when the software delays exist.

The differences in performance of the first and second iteration of the demonstrator proved that the software framework that was used had a big impact on performance. Therefore, it can be concluded that the software framework is an important factor for consideration when creating an AFE communication emulator for a highly accurate BMS. However, as it was also proved, a software framework that did not perform well on one hardware, could be sufficient on another hardware. Consequently, hardware and software are not only important as separate factors, but also as a combination, as their functionality and performance are correlated. This phenomenon was observed but not studied in detail, why no further conclusions will be drawn.

### 6.1.4 Demonstrator verification

Regarding the fact that the CSB master was used for verification of the emulator, instead of the real BMU, was considered enough for verification and proof-of-concept. The desired behaviour could be replicated in the CSB master and the desired communication could be proven with the logic analyzer. The logic analyzer showed commands and responses equal to the sampled data, concluding that the emulator would perform satisfactory if the fixed sequence was implemented in the BMU.

## 6.2 Research Question 2

*Considering differences in the quality and functionality of the emulation of the factors identified in RQ1, what is the impact on time, quality and cost by introducing the aforementioned emulator test rig for different engineering disciplines?*

Regarding the potential impact on time, quality and cost by introducing an emulator for BMS testing, the answers from the questionnaire, presented in Section 5.4, strongly imply that it is positive, i.e. that it could save time, it could improve quality and as a consequence, it could lower costs. None of the respondents answered that an emulator test rig would not be useful, suggesting that such a testing tool would make the engineers work more efficiently.

However, it is difficult to translate the respondents estimations of how much of their working hours could be used more efficiently into concrete hours, since *more efficiently* can lead to separate interpretations by different people. One would have to conduct a study comparing the engineers work without the test rig and with a test rig in order to present numbers, alternatively, conduct a more thorough analysis of different test cases. Nonetheless, the fact that 88% of the respondents answers 25% or more, could suggest that it would have a significant impact, saving weeks of work per project. When asked how it could save them time, it was also suggested that not only would the emulator cut development time, but as it would enable more testing, it could also have a positive impact on the quality of the delivered end product, meaning less time and costs would be spent on maintenance of the products.

When the respondents were asked to choose between an emulator and a commercial HIL rig, as much as 83.3% would choose the emulator. This could suggest that an emulator would be a better tool for a large part of BMS testing than the commercial HIL rig. Through the question about cases where the emulator would be too limited, it is evident that the emulator cannot completely replace the current test setups, but rather be a complement.

Regarding the impact of an emulator fulfilling the same requirements as the demonstrator, compared to a more complex emulator with more features, there were mixed opinions. Reflecting, again, that the test coverage enabled by the simpler version emulator is by no means complete.

In conclusion, this study confirms what previous research on emulation for BMS testing concludes, presented in Section 2.2, namely that an emulator test rig can save a significant amount of time, improve quality and reduce costs. That conclusion is valid for all of the studied engineering disciplines, since the splits between respondents belonging to *Software* and *Other* in regards to job title and team was close to 50/50. Supplementary to this conclusion is the fact that the software engineers rating of factors significantly differed from the other engineers, which could suggest that the requirements for an emulator to have a great positive impact on efficiency may differ between groups and that it is not certain that the one-size-fits-all principle applies. Further research would have to be conducted in order to balance requirements against each other to find the optimal effort-impact ratio.

## 6.3 Ethical considerations

Throughout the study, *the Twelve Golden Rules to Ethical Research Conduct* as well as KTH general guidelines have been followed, addressing: plagiarism, integrity of people, honoring informed consent with research subjects, and prevention of making technology openly available for misuse [35, 36]. The confidentiality of the case company Northvolt and the anonymity of the subjected employees has been respected. Any information covered by confidentiality has intentionally been left out, and when the thesis required, been modified to such an extent that confidentiality is kept, but still allowing the same representation. All questionnaire respondents were informed about the purpose of the study.

There are no safety concerns in relation to the emulator as it is a small MCU that does not handle high voltages, contrary to real batteries and HIL rigs. Testing with an emulator could therefore increase safety for the developers. Furthermore, increased testing of BMSs may improve the safety of the end products, hence improving the safety for people in the surroundings of the battery systems in use.

## 6.4 Sustainability

The most relevant sustainability aspect of the development of an emulator test rigs for BMS development is the reduction of consuming valuable LIBs for testing purposes, thus positively impacting environmental sustainability. The amount of components and materials used for developing, using and maintaining emulators is significantly less than for batteries and commercial HIL rigs. In a broader perspective, the study was focused on a company that invests not only in battery systems, but also in battery recycling. Any technology that improves the development within such a company supports the United Nations Sustainable Development Goal 7: *To ensure access to affordable, reliable, sustainable and modern energy for all* [37]. Furthermore, Northvolt's commitment to sustainability is summarized as *"[...] We recognize our responsibility as a global corporate citizen, and will work with employees, investors, partners, customers and communities to ensure that our products shift societies towards the achievement of the UN Sustainable Development Goals"* [38].

## 6.5 Recommendations to the Company

Having the BMU sending commands in a fixed sequence could have many advantages, not only for the sake of making emulator development easier, but it could also potentially improve the over all BMS quality. A fixed sequence is deterministic and predictable, meaning it could be easier to maintain, debug and analyse. As discussed previously, robust and repeatable timing of communication signals is highly critical, and a fixed sequence of commands and responses could facilitate this issue. Though the impact on functional safety by introducing a fixed sequence would have to be investigated further before implementation.

Maintaining and developing a BMS could be easier with a fixed sequence structure, as the software design would be simpler in comparison with a non-fixed sequence. Moreover, it would enable debugging using a logic analyzer, since it would not be difficult to compare the predicted behavior with the actual behavior. Additionally, a fixed sequence would enable integrating the emulator developed as a testing tool, that could be used simultaneously by many engineers in parallel.

An emulator working without fixed sequence would be more complex and would require further research and development time before it could be put to use at Northvolt. However, the many benefits that comes with using an emulator in BMS development as apparent through the Questionnaire Analysis in Chapter 5.4, makes it worth considering investing more time into emulator development, regardless if it is for a fixed sequence BMU or non-fixed sequence BMU. It is therefore in the company Northvolt's interest to encourage continuing the project with the suggestions stated in Section 6.6, Future Work.

## 6.6 Future Work

Further research could be conducted in order to investigate possible improvements and solutions for the timing issues observed in the implementation of the emulator, apart from the one briefly described in Section 5.3. Possible alternatives to explore further could be:

- To use Field-Programmable Gate Array (FPGA) instead of MCUs to speed up the software process

- Implementing new HAL functions optimized for this particular application
- Adding re-synchronisation functions for handling if the communication goes out of sync

Additional investigations could also be done on the hardware and software framework, comparing the effects of using other development boards and software frameworks. A more powerful MCU would enable faster execution and potentially solve the timing issues as well.

The features of the emulator have a great possibility to be extended in order to improve the test coverage enabled by an emulator test rig. Adding a dynamic cell model to the emulator, instead of emulating static cell data, could be a useful feature, as a significant demand exists across multiple engineering disciplines. Several different features that the respondents of the questionnaire suggested themselves could also be worth considering, i.e. emulation of cell balancing bits, internet connectivity, emulation of IC diagnostics and an auto reset function, as they would further improve the test coverage.

# Bibliography


[1] Brolin, M., Fahnestock, J., and Rootzén, J. *Industry's Electrification and Role in the Future Electricity System: A Strategic Innovation Agenda*. 2017.

[2] Croguennec, L. and Monconduit, L. *Prospects For Li-ion Batteries And Emerging Energy Electrochemical Systems*. World Scientific Publishing Co Pte Ltd, 2018, pp. 1–10. ISBN: 9789813228139.

[3] Davide, Andrea. "Custom BMS Design". eng. In: *Battery Management Systems for Large Lithium-Ion Battery Packs*. Artech House, 2010, pp. 1–14. ISBN: 1608071049.

[4] Barreras, J., Fleischer, C., Christensen, A., Swierczynski, M., Schaltz, E., Andreasen, S. J., and Sauer, D. "An Advanced HIL Simulation Battery Model for Battery Management System Testing". In: *IEEE Transactions on Industry Applications* 52 (2016), pp. 5086–5099.

[5] Munson, J. *Enabling robust data communications within a high voltage BMS*. URL: https://www.newelectronics.co.uk/electronics-technology/enabling-robust-data-communications-within-a-high-voltage-bms/143517/. (accessed: 20.11.2020).

[6] Techopedia. *Emulation*. URL: https://www.techopedia.com/definition/4787/emulation. (accessed: 26.02.2021).

[7] Pavić, I., Beus, M., Bobanac, V., and Pandžić, H. "Decentralized Master-Slave Communication and Control Architecture of a Battery Swapping Station". In: *2018 IEEE International Conference on Environment and Electrical Engineering and 2018 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I CPS Europe)*. 2018, pp. 1–6. DOI: 10.1109/EEEIC.2018.8494601.

[8] Messier, P., LeBel, Félix-Antoine, Rouleau, Jasmin, and Trovão, J. "Multi-Cell Emulation for Battery Management System Validation". In: *2018 IEEE Vehicle Power and Propulsion Conference (VPPC)* (2018), pp. 1–6.

[9] Codrey Electronics. *What is Serial Communication and How it works?* URL: https://www.codrey.com/embedded-systems/serial-communication-basics/. (accessed: 26.02.2021).

[10] Kugelstadt, T. "Extending the SPI bus for long-distance communication". In: *Analog Applications Journal* (2011).

[11] Frenzel, L. "What's The Difference Between The RS-232 And RS-485 Serial Interfaces?" In: *Electronic Design* (2013), p. 1.

[12] National Instruments Corp. *Controller Area Network (CAN) Overview*. URL: https://www.ni.com/sv-se/innovations/white-papers/06/controller-area-network--can--overview.html. (accessed: 26.02.2021).

[13] Valdez, J. and Becker, J. "Understanding the I$^2$C Bus". In: *Texas Instruments Application Report* (2015), p. 1.

[14] Total Phase. *I$^2$C Communication Background*. URL: https://www.totalphase.com/support/articles/200349156-I2C-Background. (accessed: 23.02.2021).

[15] Almendra, O. *I$^2$C vs SPI*. URL: https://aticleworld.com/difference-between-i2c-and-spi/. (accessed: 23.02.2021).

[16] RF Wireless World. *What is UART?* URL: https://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-UART.html. (accessed: 23.02.2021).

[17] FTDI. "What is CAN?" In: *Future Technology Devices International Limited Technical Note* (2015), pp. 5–11.

[18] Total Phase. *What is CAN Bus Protocol?* URL: https://www.totalphase.com/blog/2019/08/5-advantages-of-can-bus-protocol/. (accessed: 23.02.2021).

[19] Cartecc. *CAN-Bus*. URL: https://www.kfz-tech.de/Engl/Biblio/Digitaltechnik/CAN-Bus.htm. (accessed: 23.02.2021).

[20] RF Wireless World. *What is RJ-485*. URL: https://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-RS485.html. (accessed: 23.02.2021).

[21] Microlink Engineering Solutions. *Technical Notes: Communication Links*. URL: http://www.microlink.co.uk/link.html. (accessed: 24.02.2021).

[22] AdvanTech. *What is RS-232? Fundamentals of the Protocol*. URL: https://www.advantech.com/resources/white-papers/506c05a1-f599-4615-ac54-8259385d61e8. (accessed: 15.04.2021).

[23] EL-PRO-CUS. *Difference Between RS232 and RS485: Advantages and Disadvantages*. URL: https://www.elprocus.com/difference-between-rs232-and-rs485-advantages-and-disadvantages/. (accessed: 23.02.2021).

[24] FingerTec. "The Disadvantages of RS485 Communication Method". In: *TCP/IP Network as Main Communication in Electronic Access Control Systems* (2013), p. 1.

[25] Bishop, R. H. *Mechatronics: An introduction*. 1st ed. Taylor and Francis Group, LLC, 2015, pp. 2.9–2.14. ISBN: 9780849363580.

[26] Buccolini, Luca, Orcioni, S., Longhi, S., and Conti, M. "Cell Battery Emulator for Hardware-in-the-Loop BMS Test". In: *2018 IEEE International Conference on Environment and Electrical Engineering and 2018 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe)* (2018), pp. 1–5.

[27] Collet, A., Crébier, J., and Chureau, A. "Multi-cell battery emulator for advanced battery management system benchmarking". In: *2011 IEEE International Symposium on Industrial Electronics*. 2011, pp. 1093–1099. DOI: 10.1109/ISIE.2011.5984312.

[28] Håkansson, A. "Portal of Research Methods and Methodologies for Research Projects and Degree Projects". In: *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering FECS'13* (2013), pp. 67–73.

[29] IKALOGIC. *SP209 Series Logic Analyzer*. URL: https://ikalogic.com/sp209-logic-analyzer/. (accessed: 25.02.2021).

[30] IKALOGIC. *ScanaStudio*. URL: https://ikalogic.com/scanastudio/. (accessed: 25.02.2021).

[31] Airbrake. *V-Model: What Is It And How Do You Use It?* URL: https://airbrake.io/blog/sdlc/v-model. (accessed: 05.01.2021).

[32] Innovative Testers. *Software testing: V-model*. URL: https://innovativetester.wordpress.com/software-testing/introduction-to-software-testing/v-model/. (accessed: 29.03.2021).

[33] arm Mbed OS. *NUCLEO-F103RB*. URL: https://os.mbed.com/platforms/ST-Nucleo-F103RB/. (accessed: 03.03.2021).

[34] arm Mbed OS. *NUCLEO-F746ZG*. URL: https://os.mbed.com/platforms/ST-Nucleo-F746ZG/. (accessed: 03.03.2020).

[35] European Commission. *Ethics for researchers*. Luxembourg: Publications Office of the European Union, 2013. ISBN: 978-92-79-28854-8.

[36] Department of Machine Design. *Master's Thesis Project at the Department of Machine Design, Guidelines for 2020/21*. KTH School of Industrial Engineering and Management, 2020.

[37] United Nations, Department of Economic and Social Affairs. *Sustainabile Development Goal 7*. URL: https://sdgs.un.org/goals/goal7. (accessed: 27.04.2021).

[38] Northvolt. *A Commitment to Sustainability*. URL: https://northvolt.com/about. (accessed: 27.04.2021).

# Appendix - Contents

# Appendix A

# Emulator

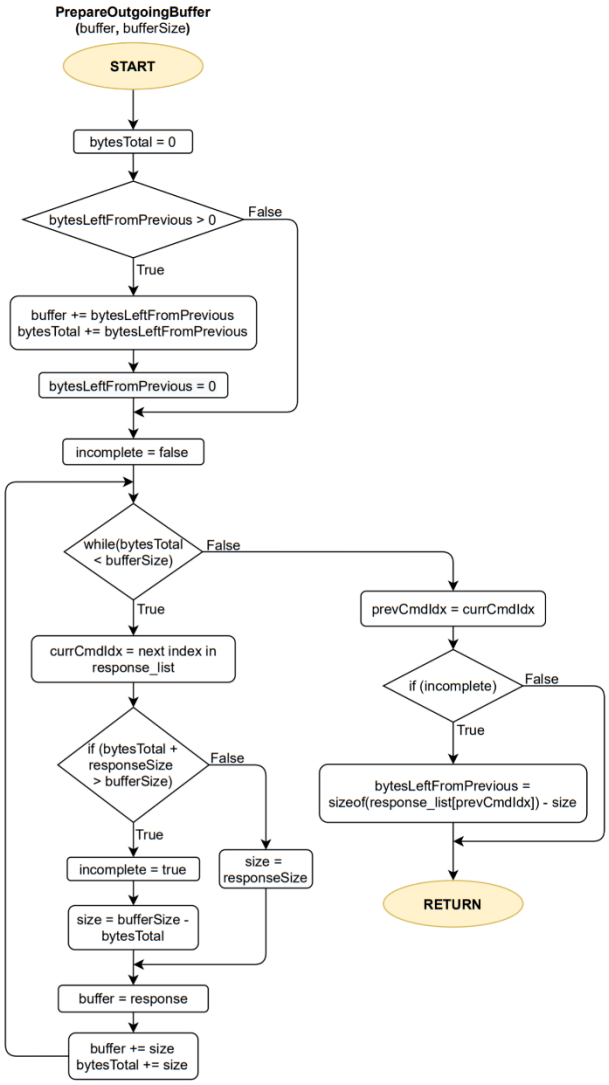



Figure A.0.1: A flowchart of the function `PrepareOutgoingBuffer`.

# Appendix B

# Questionnaire

# Master Thesis Project - Survey

What is this?
We, Miko and Julia, are doing a project within our Mechatronics engineering education at KTH Royal Institute of Technology, in collaboration with Northvolt Battery Systems. We need your help to evaluate our work so far!

Who are you?
This survey is for engineers at Northvolt Battery Systems working with BMS testing or related activities (preferably HIL rig or commissioning), at least half a day per week, and who have at least half a year's experience.

This survey will take about 10-15 minutes of your time.

* Required

## We want to make BMS testing easier!

The goal of the project is to create a tool that could be used for BMS testing by replacing the battery + Cell Sensor Board (CSB) interfacing with the BMU. We call it an emulator since it emulates, or imitates, the behavior of the CSBs.

So, instead of working with the subpack setup or a setup with the HIL rig, you could also have the option to work with a setup with the emulator. See Figure 1 and 2 below for an overview. The block "Emulator" in Figure 2 is all software running on an MCU.

### Current setup VS. Setup with Emulator

Figure 1

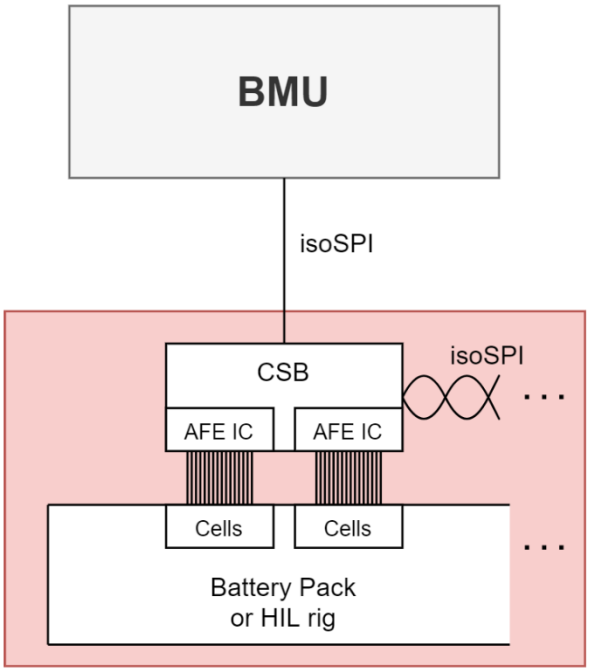
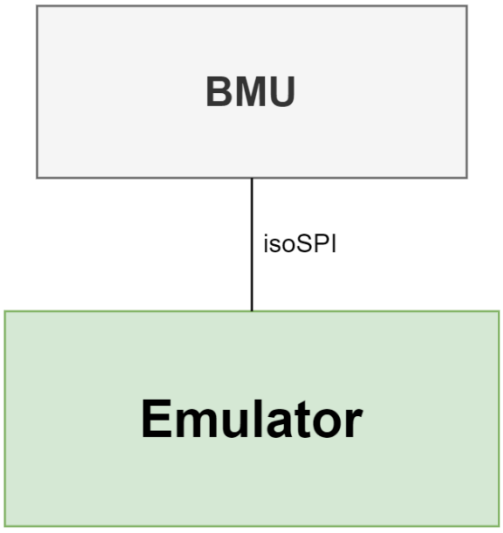



Figure 2



## What is the emulator and what does it do?

The emulator:
* Would mainly emulate the CSB(s) that are measuring cell parameters such as cell voltage and temperature.
* Has the same isoSPI interface as the real CSB and it is configured as a communication slave just as the real CSB.

## What is this survey about?

The questions of this survey will be about what properties you think are important for that emulator to be useful for you, and also how your work could become easier (for example less configuration time and troubleshooting).

### Personal questions

1. What is your job title? *

_____

2. What team are you a part of? *

*Mark only one oval.*

- ◯ BMS, Software
- ◯ BMS, Hardware
- ◯ BMS, System
- ◯ BMS, Application
- ◯ BMS, Quality Assurance
- ◯ Connected Batteries, Software
- ◯ Other: _____

3. How much experience do you have with BMS testing? *

*Mark only one oval.*

- ◯ 1/2-1 year
- ◯ 1-2 years
- ◯ 2-5 years
- ◯ 5+ years

## Emulator questions

0 = not important at all, 5 = very important

4. {HIGH-LEVEL} How important are these factors for an emulator to be useful for you? *

*Mark only one oval per row.*

| | 0 | 1 | 2 | 3 | 4 | 5 | Not sure |
|---|---|---|---|---|---|---|---|
| Cell voltages and temperatures can be manually set to values of your choice | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ |
| Cell voltage and temperature measurements are not constant values, but have a realistic behaviour (drive cycle) | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ |
| Flexibility: The number of CSB:s can be configured as you want | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ |
| User-friendliness: Easy to setup and run the emulator (less than 15 min to set up) | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ |

5. Optional: {LOWER-LEVEL} How important are these factors for an emulator to be useful for you?

*Mark only one oval per row.*

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| The emulator verifies the received command from the BMU by checking the PEC | | | | | | |
| The emulator replies to Read Status Register command | | | | | | |
| It's possible to clear the Cell/Aux/Status Register groups | | | | | | |
| It's possible to write values to the emulated memory (Write Configuration command) | | | | | | |
| The emulator replies to Read Configuration command | | | | | | |
| The emulator replies to Poll ADC Conversion Status | | | | | | |

6. Optional: other important features not mentioned above? Please rate them on a scale 0-5

7. Consider an emulator that can do all the things listed in the previous questions. How useful would it be to you? *

*Mark only one oval.*

- ◯ Not useful at all
- ◯ It could be useful occasionally
- ◯ Very useful for me

8. Please estimate how much of your working hours could be used more efficiently if the emulator was available? Choose the one closest to your estimation *

*Mark only one oval.*

- ◯ 0%
- ◯ 25%
- ◯ 50%
- ◯ 75%
- ◯ 100%

9. Considering time, quality and costs: Which of the following two would you choose to have at hand in your work? *

*Mark only one oval.*

- ◯ An emulator fulfilling above requirements (this is the cheaper, more scalable choice with a simple cell model, easy to setup and configure)
- ◯ The commercial HIL rig (expensive, not scalable, risks for HW issues hard to troubleshoot, ovely complex for some tests)
- ◯ None of these would add any value to my work

## Open questions

Consider an emulator fulfilling the following:
* Interfaces with a BMU through isoSPI
* Replies correctly to read commands from the BMU
* Configurable number of emulated CSB:s
* Emulates cells as having constant voltages and temperatures
* Constant status and configuration registers

10. How, if at all, would having such an emulator save you time in your work with BMS testing? *

11. Could you give at least one example of a case where the emulator would be too limited for BMS testing? (Eg. too limited cell model, does not enable testing of the real CSB.) *

**Please, consider an ideal emulator that doesn't have the limitations stated previously.**

It includes all fidelity of a real CSB, e.g. the internal state machines, logic, read/write registers, etc.

12. What difference do you think a full emulation would make in your work compared to the previous limited CSB emulator?

Thank you so much for your time!

**We look forward to see you at our thesis presentation at the end of May 😊**

# Appendix C

# Rating of factors

The boxes in the box plots show the interval capturing half of the responses. Outliers are marked with circles, and lines mark the intervals where 95% of the responses are. The orange line found in every box shows the median of every groups set of answers.

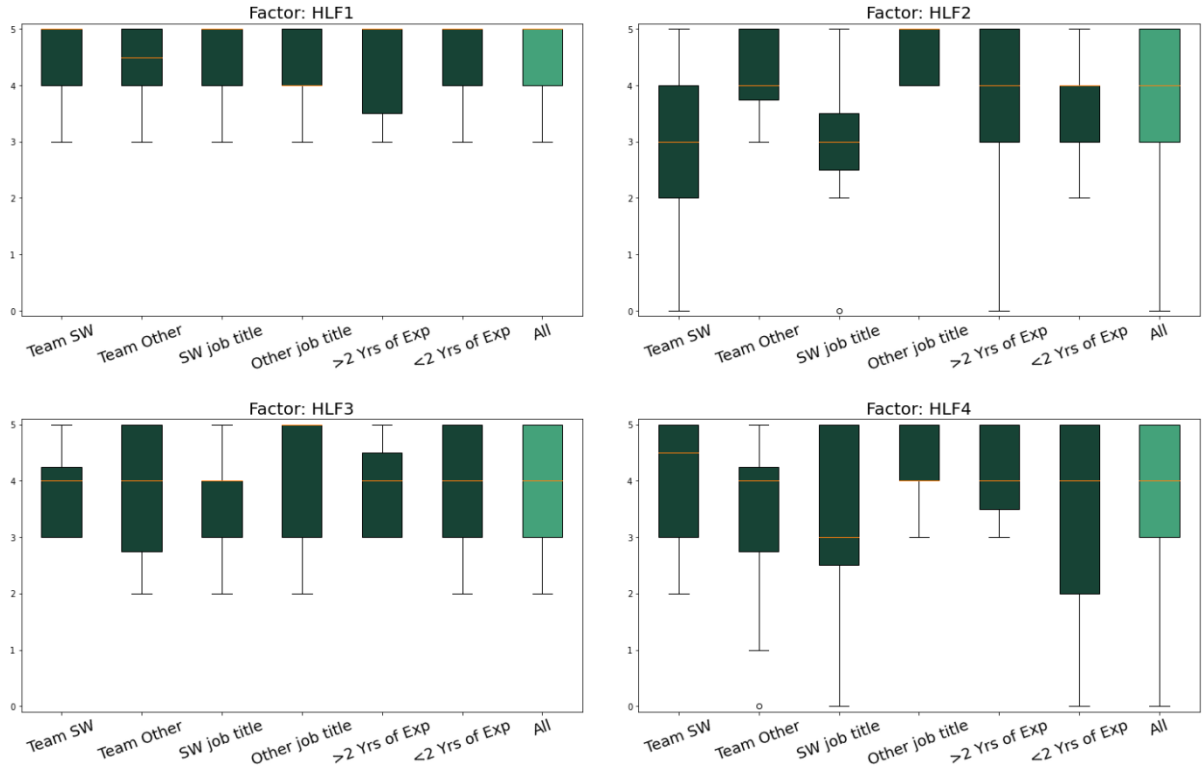



Figure C.0.1: Each group's ratings of the importance of four high-level factors.

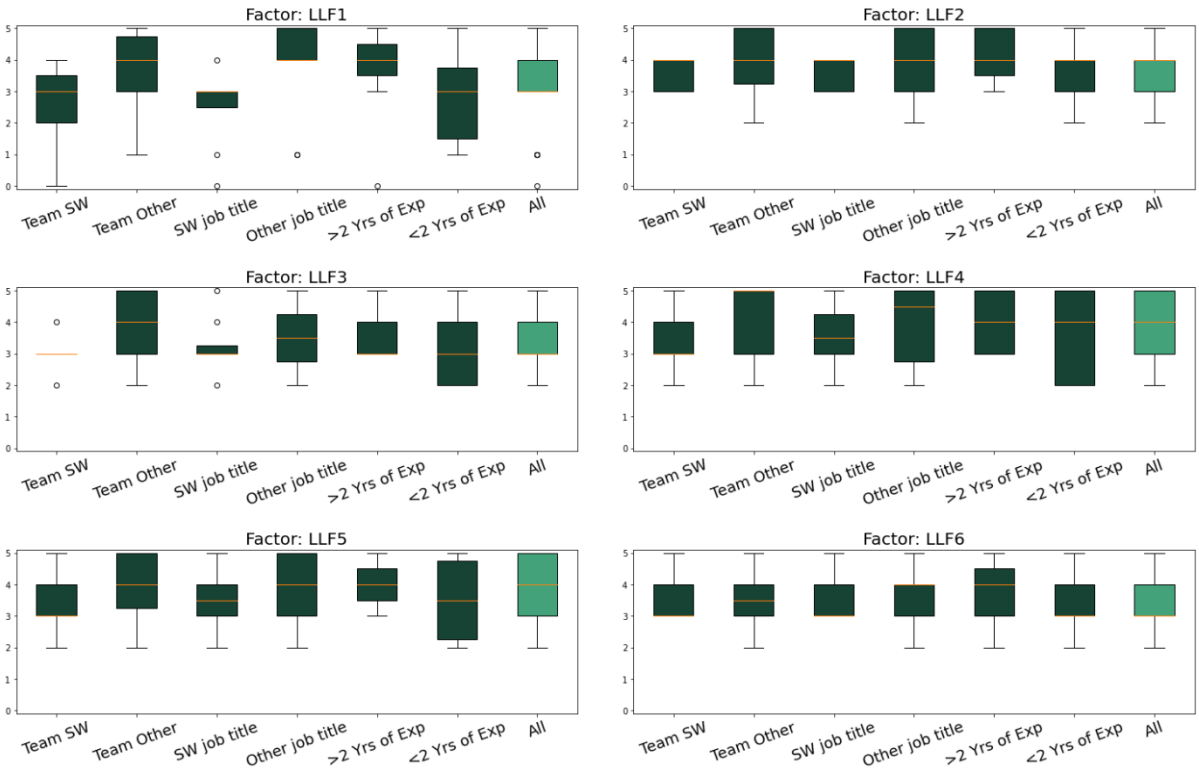



Figure C.0.2: Each group's ratings of the importance of six low-level factors.

TRITA ITM-EX 2021:106