POLITECNICO DI TORINO

Dipartimento di Elettronica e Telecomunicazioni

Analysis and Modeling of an Energy Management System in Automotive Environment

Tesi di Laurea Magistrale in
Ingegneria Elettronica

Relatore:
Prof. M.R. Casu

Candidato:
Leandro Malara

Marzo 2020

Contents

List of Figures

Chapter 1

Introduction

In the pursue of less CO_2 emissions and pollution, the last decade saw an exponential growth of batteries driven applications. This growth evolved in many directions: it involved classic vehicles for public roads (cars, buses, vans, etc.), competition vehicles in tracks, but also industrial equipment and infrastructures (e.g. backup power for elevators), and gave birth to new micro-mobility concepts and new vision for public transportation. Thus the need for a system that manages all battery pack operations and its safety became evident. The requirement for this system may vary depending on the target application and use case. In this document I put the focus on what is the modern automotive energy storage use case, that may be also valid for the next years to come, until new technologies, currently developing at early stages, incompatible with the current ones in terms of system management, will be available.

The objective of the thesis is to analyse the requirement of a system capable of handling a typical automotive battery pack and model it, both in hardware and in software, in a prototypal version, on a microcontroller platform with existing hardware. The prototype has been tested on a test bench with evaluation modules and a small battery string. This involved analysing a broad spectrum of aspects related to batteries itself but also to their use inside battery packs. An introduction of some concepts and topics will also be given in order to better understand the document. The carrying out of my work has been done in collaboration with Bylogix, inside their facilities.

Chapter 2

Energy storage systems introduction and analysis

The need of a battery pack management system comes from two contributions: one is the cell itself, in order to guarantee that they will operate in optimal conditions in terms of safety and performance; the other is the environment of interaction, i.e. the battery pack subsystem and the other subsystems of the vehicle. For this reason in this chapter it will be given an introduction on current battery cells technologies, in order to identify their problems, advantages, disadvantages, and the parameters which need to be monitored. An analysis on the typical structure of a battery pack and its implementation inside a vehicle architecture will determine the feature and characteristics that the management system should meet.

2.1 Batteries principles

The most known electrochemical devices that allows to store energy through it are:

- **Rechargeable batteries**: also called *accumulators*, allows for energy to be charged and discharge, theoretically infinite times, composed of an anode, a cathode and an electrolyte. Depending on the materials used for these parts, the resulting battery may have different advantages and disadvantages in terms of energy density, specific density, power density and safety risks. The resulting nominal voltage across the terminals is ranging between 1.2V and 3.9V
- **Supercapacitors**: constituted by two electrodes separated by an ion permeable membrane, ionically connected by an electrolyte, as opposed to having a solid dielectric between them like in conventional double layer capacitor. It is an hybrid between a double layer capacitor and a rechargeable battery, having a power density far higher than the latter but still lower than the former.

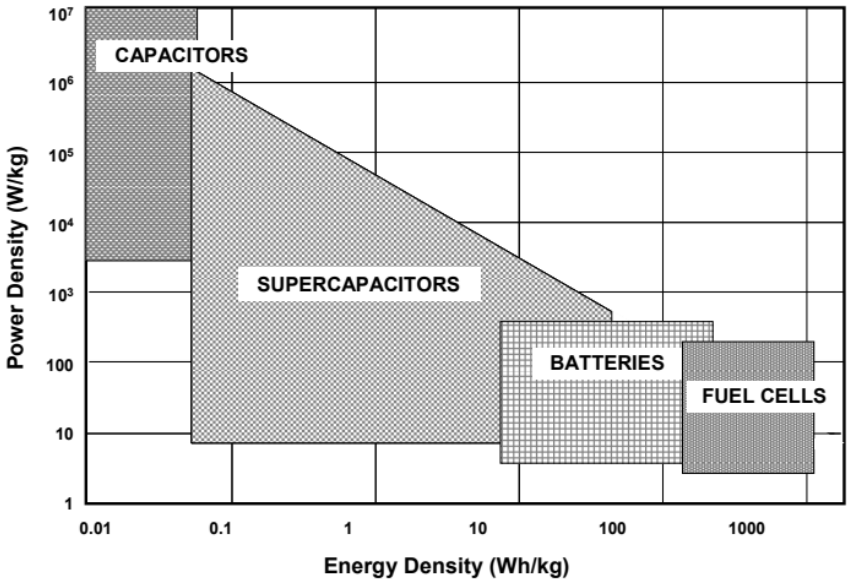

Figure 2.1: Power vs Energy density chart (Ragone plot)

It is noticeable, as reported in [4] and shown in figure 2.1, how supercapacitors and rechargeable batteries have a wide range of possible characteristics, and thus

application field. Although there exist some application of supercapacitors in the automotive industry, they are mostly related to high performance vehicles as secondary traction energy source, like in the case of Lamborghini latest supercar, the *Sian*, or to public transportation like buses or trams. Instead, almost every production car with a battery, whether it is used for traction or supplying energy for all other services, relies on rechargeable batteries.
The most commonly used, due to their technological advancement and cost are:

- **Lead-Acid**
- **Lithium Batteries (LIBs)**, which comprehends among the others Lithium Cobalt Oxide, Lithium Nichel Aluminum Cobalt Oxide, Lithium Iron Phosphate and many others
- **Nickel-Metal Hydride (NiMH)**

Also, it must be noted that LIBs can assume many forms, which have been recently stardandized by various international organization, like the International Organization for Standardization (ISO), regarding their size. In particular, Prismatic, Pouch and Cylindrical are the common standard type cells. With the exception of Toyota, which made a large long term investment on NiMH, all recent car makers uses LIBs due to their large saving in weight and space. Since they are the most technologically advanced and it is expected an increase in overall performance in the next decade, as targeted by national organization like USABC, NEDO, EUCAR (USA, Japan, Europe), this will be the chosen type of battery that this project will be based on.
As said at the beginning of the chapter, there are three main parameters to be monitored in a cell: temperature, voltage, current output. These are crucial in performance improvement and safety of the batteries, because LIBs, more than other rechargeable battery types, have a quite narrow window of safe operation zone, which if exceeded can lead to permanent loss of performance or, in the worst case, to thermal runaway and subsequent explosion or fire. It is deduced then that the management system should perform a double task, guaranteeing safety operation and improving performance by tracking these characteristics and acting on external environment.

2.2 Battery packs: overview

Since obviously a single cell would not provide sufficient energy, they are collected in strings of series and parallel cells. Their number is highly depending on the target application, on which we can have an overview:

- **HEV**: the Hybrid Electric Vehicless, being at the moment the most diffused type of hybrid vehicle, uses small energy battery packs (0.7∼1.6 kWh) usually with a single string of series cell. Their battery can't be externally charged, and it is used for accumulating energy during braking and releasing it in the form of small power boost, which can be up to 60 kW in the case of highly performing vehicles. These vehicles can't be powered only by the electric powertrain, but rather assisted by it

- **PHEV**: the substantial difference from Plug-In Hybrid Electric Vehicless and HEV is that the battery can be externally charged. This time the energy can range typically from 3 to 15 kWh, with some exceptions reaching 30kWh, as in *Porsche 918* car, and the battery usually is made of more strings of cell in parallel. The level of integration with the Internal Combustion Engine (ICE) is deeper, providing the capability of covering up to 60km while being powered only with the electric motor(s)

- **EV**: in Electric Vehicles, the only power that can move the vehicle comes from the battery. Battery energy span is very large (30∼100 kWh), and consists in a high number of parallel strings.

The terminology that will be used for indicating the cells arrangement in a battery pack is $X\mathrm{S}Y\mathrm{P}$, which states for Y parallel strings of X series.
In addition, when considering the type of battery pack, another parameter must be accounted for: its working voltage. As defined in [5], there is a classification between Low Voltage (LV) ($\leq$ 60V DC or 30V AC) and High Voltage (HV) ($\geq$ 60V DC or 30V AC, and $\leq$ 1500V DC or 1000V AC), called respectively voltage class A and B by International Organization for Standardization (ISO) in [6]. Few HEV

implements a LV architecture, due to the lower cost, because when dealing with HV systems, more precautions need to be taken both in electronic circuits and mechanical components. Instead virtually every PHEV and EV vehicle on the market is based on an HV system, with a nominal voltage of $\sim 400V \pm 50V$, but exceptions exist, like the latest *Porsche Taycan* with 800V.
For the requirement of the proposed system, I will focus my analysis in series "dominated" battery packs, because parallel strings does not add complexity in terms of management. The battery pack is of course made from strings of cells, but that is not the only component. A definition of a battery pack in [7] says that it is made from:

- Voltage class A connections
- Voltage class B circuit and connections
- Cell assemblies
- Cell electronics
- Cooling interface
- Service disconnect
- Battery Control Unit (BCU)
- inside an impact-resistant case

;oreover, the definition of BCU is an object that calculates State-of-Charge (SoC) and State-of-Health (SoH) and provides battery operational limits to the vehicle management unit. It may have direct access to the main contactors of the battery system in order to interrupt the voltage class B circuit under specified conditions, e.g. overcurrent, over voltage, low voltage, high temperature, and may vary in design and implementation (i.e. integrated in a single or multiple unit(s) and placed inside or outside the battery pack). Moreover, some of its funcionalities may be implemented by other vehicle control units [7]. All of the previously cited components making up the battery pack will be analysed in the next chapter in order to set constraints

to meet the requirements in the ISO rule. Notice that the ISO rules have not been used as a target, as it would not be possible for a project like this, but rather as a criteria and guidance.

Chapter 3

Slave unit modeling

The first subject to treat is how to measure and monitor the cells, the part of the battery pack previously called "cell electronics". As said, there are two possibilities for the management system: a distributed system and a centralized one. If we take a look to every battery pack, the total of the cells is divided into (usually equal) battery segments for more reasons: cheaper production mechanism, easier assembly and intervention for post production (lower weight) and lower voltage at the terminals of each segment (the division is made by breaking the series string). Due to this segmentation, a centralized unit would not be feasible for large battery packs: connections would be too long and too many, with a large and expensive printed circuit board (PCB), and it would not be as modular as a decentralized unit.

An option could be to have replicated hardware units doing the same thing on each segment, and a central unit collecting all the information. This solution is often referred as a *Master/Slave* configuration. The focus now will be on the *Slave* unit. The main task for this unit will be measuring cell temperatures/voltages, and share the information with the *Master* unit. This can be obtained in different ways, two of them will be analysed.

3.1 Microcontroller approach

The first way to implement the *Slave* unit would be with a microcontroller and a suitable ADC interface with the cells. When measuring a string of cells, which voltage can go up to several tens of volts, with a simple microcontroller, some architecture possibilities are proposed in the next figures. Shown in 3.1, the simplest approach would be to choose suitable resistors to make voltage dividers. The ADC peripheral of the μC has a the lowest cell negative terminal as lower voltage reference. Although simple, voltage dividers would result in poor uncertainty due to resistors intrinsic uncertainty, on the other hand the use of high precision resistors would increase cost. Also the maximum number of measurable cells depends on the chosen μC ADC input lines, which is usually around 8 for a 48 pin package. Otherwise an external A2D with more input channels should be used.

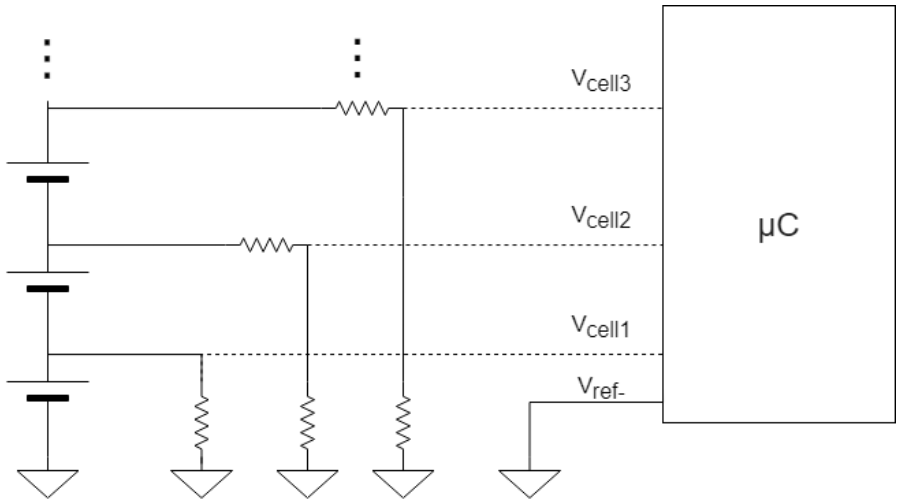

Figure 3.1: A2D with resistor dividers approach

Another approach [3.2] is to make the reference voltage changing with every measured cell. This can be achieved by using digital bilateral switches like the *CD4066B* or similar, and alternating the control signals in order to measure all cells. Just one ADC input channel could be used, but one control signal is needed for each cell, and switches integrated circuits could be expensive.

The most difficult task however would be implementing the communication between the *Master* and *Slaves*. At the hardware level, two simple solutions could

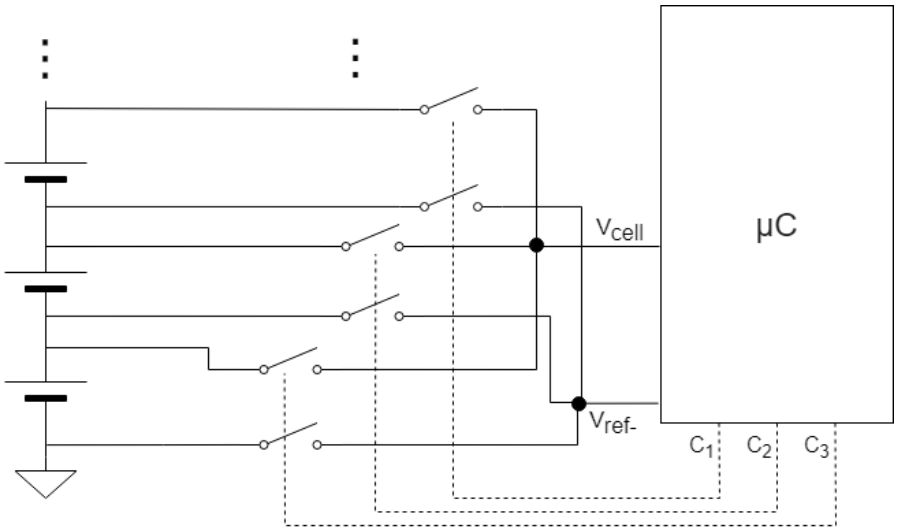

Figure 3.2: A2D with switches

be:

- **I2C**: a single I2C bus has the benefit of using only two wires and 7-bit addressing is enough for the number of slave units we are dealing with.
- **SPI**: can be used in *daisy-chain* configuration, because otherwise it would lead to an high number of *chip-select* lines.

Both buses can be physically deployed as a chain by using intermediate board as buffers for the next boards [3.3] , leading to a simple layout.

These solutions will not be further detailed as the common problematic factor would be designing the communication protocol. Tuning the synchronization between all the *Slaves* with respect to each other and to the *Master* could be a difficult task. Every *Slave* unit needs to be singularly programmed and their position in the chain is crucial, meaning for example that they can't be swapped or replaced with ease with another one.

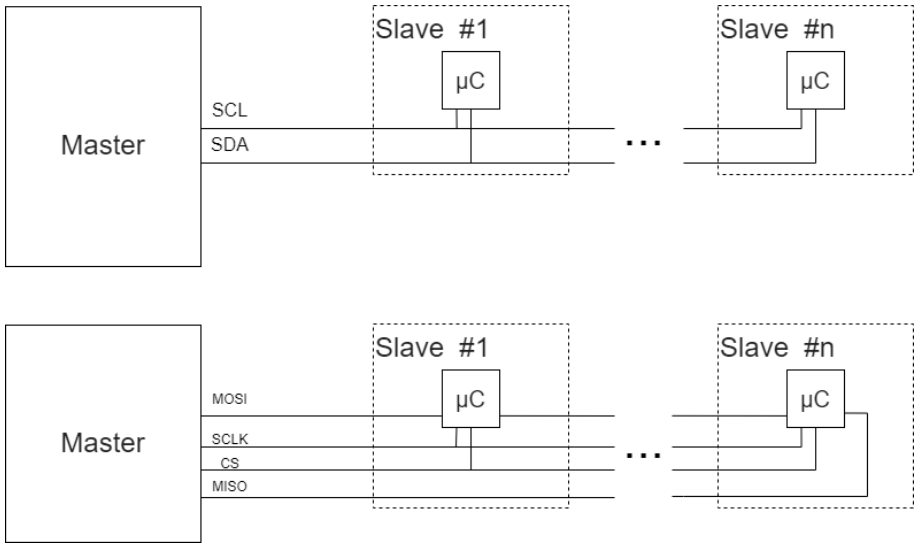

Figure 3.3: I2C (top) and SPI in daisy-chain (bottom) layouts

3.2 Final choice and implementation

Looking for a commercial available solution can be the best sometimes. At the beginning of portable electronics era, relying on a single battery cell, some integrated circuit (IC) were deployed, able to measure and monitor a single cell, in a more or less independent fashion. As of today, with growing applications relying on a string of cells, many products developed under the form of *multi-cell monitoring* IC, differentiating for number of possible cell to be monitored, communication protocol, cell balancing capability, auxiliary I/O, and many other peculiarities. An example are *Analog Devices LTC68XX* and *Texas Instruments BQ76XXX* series. The chosen IC that will serve as *Slave* unit is the *Texas Instruments BQ76PL455* from the latter family. In this section it will be discussed how this has been used in the final implementation, with detailed description on hardware needed for correct functioning.

3.2.1 Basic Principles

This device can measure from 6 to 16 cells connected in series, can be stacked in a chain of up to 16 devices, allowing to monitor a battery pack that can reach the limit of voltage B class. The communication in the chain happens with a capacitor-isolated serial transmission (similar to a Universal Asynchronous Receiver Transmitter (UART)). A μC host is needed, that can communicate only with the bottom device of the chain with a normal UART protocol. It has a 14-bit Successive Approximation Register (SAR) ADC measuring the cells and an additional 8 auxiliary analog inputs that can be used for temperature monitoring or other sensors. Passive balancing can be activated by dedicated pins. The device is powered from the cells portion that it is monitoring.

3.2.2 Cell measurement

The IC is able to measure up to 16 cells, from a minimum of 6. This is achieved by adopting a level shifter, lowering the voltage level of higher cells, directly integrated

in the die in a portion called Analog Front-End (AFE). The ADC input is multiplexed in order to measure not only cell voltages but also: auxiliary inputs, internal voltage reference(s) and internal temperature sensor for monitoring the device status itself. At the hardware level, the interface with the cells (reported in 3.4 for the first 2 cells is) serves different purposes. Each cell positive tap input (named Cell$n$) is fused and for voltage measurement the ADC input is filtered with a low pass RC filter of the 1st order ($R5$ and $C2$). Resistor is also used for limiting inrush current to the pin in case of hot-plug.

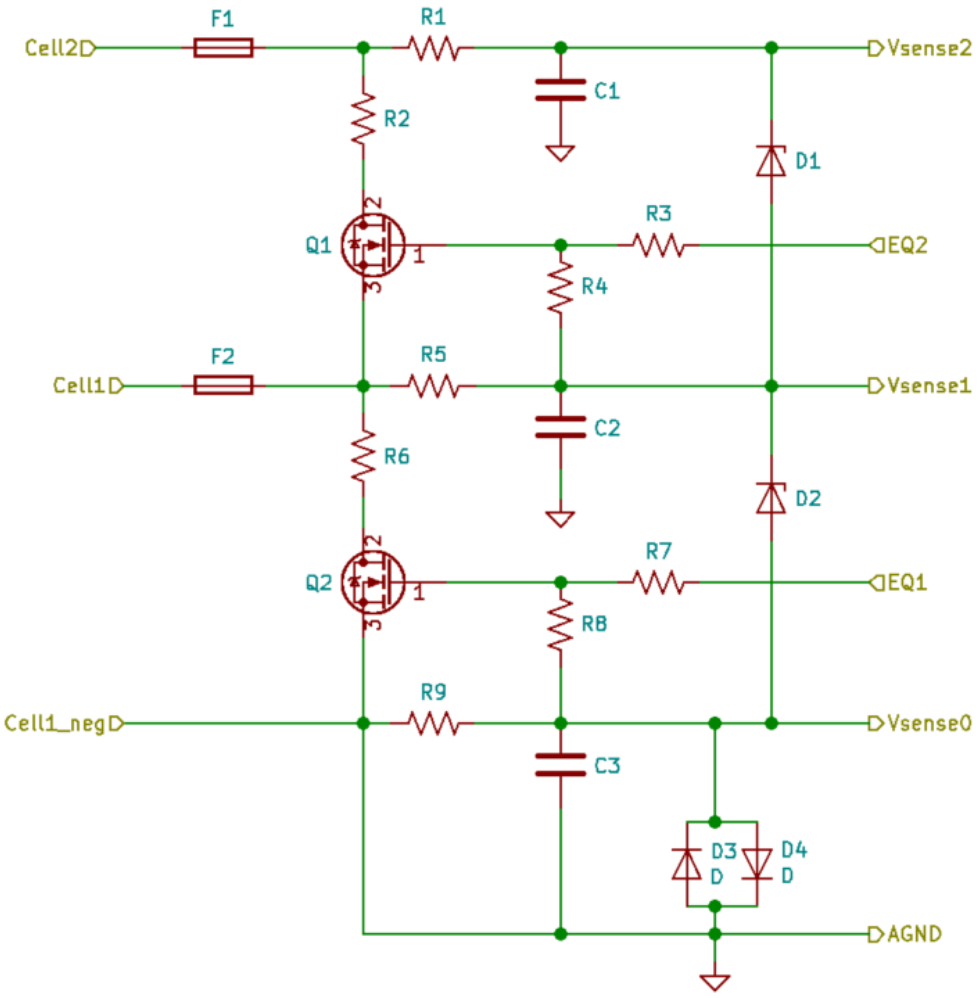

Figure 3.4: hardware interface between BQ76 and cells

The choice of the value for resistance and capacitance depends on the target performance, but however manufacturer specify the bounds to $100\Omega \sim 1k\Omega$ for the resistance and a minimum of $0.1\mu F$ for the capacitor. In particular, the resistor

value affects the ADC error measurement by a quantity:

$$V_{err} = 2RI_{sample}$$

where $I_{sample} = 7.6\mu A$ is specified as maximum value in test conditions. Choosing a resistance of 220Ω and a capacitor of $0.22\mu F$ we have:

$$V_{err} = 3.3mV, f_{cut} = \frac{1}{2\pi RC} = 3.3kHz$$

which is a good result. Notice that all capacitors are connected to lowest cell ground, which means that in the worst case (16 cells connected) they must withstand at least 80V. Ceramic capacitors rated X7R and 100V in 0805 package are a compromise between size and cost. In addition, in order to limit transient voltages from to adjacent channels, a clamping zener diode (*D2*) of $V_z = 6.2V$ is added. Both positive terminal of top cell (V_TOP) and negative terminal of bottom cell($Vsense0$) are input to the ADC for measuring total string voltage. The connection to V_TOP [3.5] is coupled to $Vsense16$ through two back-to-back signal diodes, with fast response and high V_f, for hot-plug robustness and to avoid noise coupling to the adjacent channel. The same is applied to the connection between $Vsense0$ and the analog ground pin ($AGND$). When less than 16 cells are used per *Slave*, all vacant pin connections must be tied to the highest cell, and can be done by replacing the zener diodes with 0Ω resistors, thus the design of the board is not modified, but components selection during assembly assures modularity.

3.2.3 Cell balancing

When cells are connected in series, the energy that can be charged/discharged will depend on the conditions on the single cell: the one with highest voltage will limit the pack when charging, the one with lowest when discharging. The difference on energy/voltage level between cells in series is due to many reasons, for example storage conditions of single cells before assembly or temperature difference when in operation due to the cooling system. Achieving a balanced battery pack is fundamental in order to maximize the energy stored and at disposal for the vehicle. This

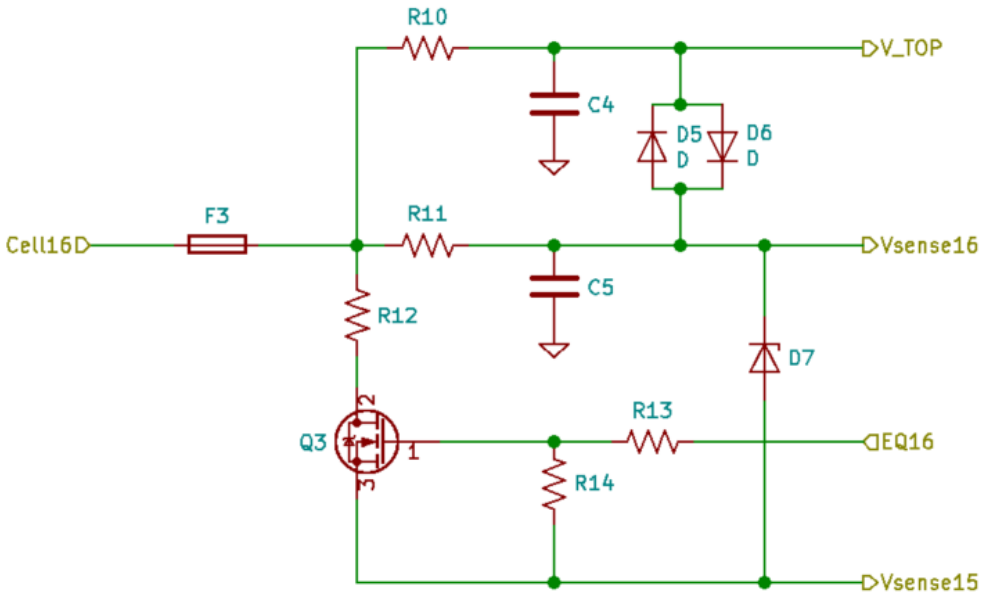

Figure 3.5: highest cell hardware interface

could be done in dissipative and non-dissipative way. The non-dissipative method uses flyback DC-DC converters in order to take energy from most charged cells and transfer it to less charged ones; its cost in terms of control system and extra hardware make this solution less desirable, but considering its main advantage, speed, it could be implemented in the perspective of future super-fast charging. The dissipative method consists in discharging individually all cells down to the level of the lowest cell. In the *BQ76PL455A* there is a control signal (EQn) able to control an external NMOS (Q2) that will discharge each cell over a resistor (R6). The choice for the resistor depends on the energy stored in the series connection, accounting for all cellls connected in parallel. This energy value will be lower for high voltage, thus high series connections, battery packs and higher for low voltage, high parallel connections (or just cells with high ampacity). For example choosing a 50Ωresistor, a typical Li-Ion cell at its maximum voltage of $\sim$ 4.2V will dissipate 0.35W, that can be obtained with a surface mounted component in 1206 package. For a large battery of let's say 60kWh and made with 100 series connections, a constant dissipation of 0.35W means being able to equalize the battery by 1% every 17 hours. In this case using larger through-hole axial resistors, e.g. 20Ω rated for 2W, this time

can decrease down to 3.5 hours. Of course in the latter case, the *Slave* board area would increase, and also heat dissipated by resistors should be taken away from the board. The FET must be chosen with $V_{DS} \geq 50V$ and $V_{GS} \geq 15V$, preferably with an integrated zener protection for the gate. In addition, resistor R7 limits current going to the EQ pin of the IC during inrush current events, resistor R8 make sure the gate is not left floating.

3.2.4 Temperature measurement

In this decentralized model, the chosen *Slave* unit is capable of measuring 8 analog additional inputs. Two types of analog sensors commonly used are thermocouples and thermistors. Thermocouples working principle is the seebeck effect, that is a voltage difference across two metals produced by the temperature gradient between a cold and hot end. This method requires knowing the temperature of the "cold" end by the use of a specific local circuit, and can be inconvenient when measuring a lot of points located far from each other. The use of thermistors is easy and cheap, with the drawback of a more limited temperature range and accuracy. A Negative Temperature Coefficient (NTC) thermistor has been used for this project, but final choice will depend on application. The chosen thermistor is from Murata, has a nominal resistance $R_0 = 10k\Omega$ @ $25°C$ with an operating range of -40°C to +125°C (see 3.6).

Part Number	Resistance (25°C) (ohm)	B-Constant (25-50°C) (K)	B-Constant (25-80°C) (Reference Value) (K)	B-Constant (25-85°C) (Reference Value) (K)	B-Constant (25-100°C) (Reference Value) (K)	Maximum Operating Current (25°C) (mA)	Rated Electric Power (25°C) (mW)	Typical Dissipation Constant (25°C) (mW/°C)
NXFT15XV302FA□B□□□	3k±1%	3936±1%	3971	3977	3988	0.22	7.5	1.5
NXFT15XH103FA□B□□□	10k ±1%	3380 ±1%	3428	3434	3455	0.12	7.5	1.5
NXFT15XV103FA□B□□□	10k ±1%	3936 ±1%	3971	3977	3988	0.12	7.5	1.5
NXFT15WB473FA□B□□□	47k ±1%	4050 ±1%	4101	4108	4131	0.06	7.5	1.5
NXFT15WF104FA□B□□□	100k ±1%	4250 ±1%	4303	4311	4334	0.04	7.5	1.5

Figure 3.6: NTC specifications, chosen one is the second

In order to be measured, the NTC can be used as a resistor in a voltage divider circuit [3.7]. The value of R15 can be chosen as the smallest possible value that assure thermistor safe operating conditions (maximum current and power dissipation) so that voltage swing is maximized, thus precision. The worst condition is at maximum temperature when the resistor value is at minimum. Since +125°C are unrealistic in

normal operating condition inside a battery pack, +80°C will be used as maximum. $R_{min} \approx 1.66k\Omega$ @ +80°C (see equation 3.1)

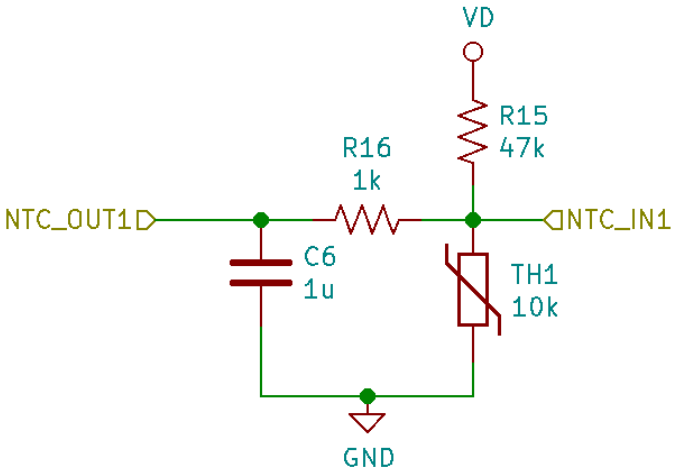

Figure 3.7: circuit for NTC measurement

$$R = R_0 * e^{\beta(\frac{1}{T} - \frac{1}{T_0})} \tag{3.1}$$

At this value, current flowing in the NTC needs to be:

$$\frac{V_{DD}}{R15 + R_{min}} \leq I_{max} = 0.12mA$$

thus $R15 \geq 43k\Omega$. Regarding lower temperatures, also here -40°C are unrealistic and -20°C will be consiered as minimum: $R_{max} \approx 27k\Omega$. If the resistor value is 47kΩ, the voltage swing will be:

$$V_{max} - V_{min} = 1.9937V - 0.1808V = 1.7529V$$

If the system should be able to have a 0.1°C precision on a 100°C temperature range, the ADC should be able to have at least $V_{res} \leq \frac{1.7529}{1000} = 1.75mV$. The 14-bit ADC with $V_{ref} = 5V$ of the *BQ76PL455A* has a voltage resolution of $V_{res} = \frac{5V}{2^{14}}$ so the requirement is met. Notice that the value of V_{DD} used as power supply for

the NTC circuit is 5.3V, as described later in the chapter. An additional low pass RC filter (R16, C6) with $f_{cut} = 160Hz$ provides some noise immunity. If the working environment is too noisy, a pseudo differential measure can be performed (3.8) by using two inputs of the ADC instead of one and subtracting the two readings to obtain the voltage across the thermistor, at the cost of halving the number of measurable sensors to 4.

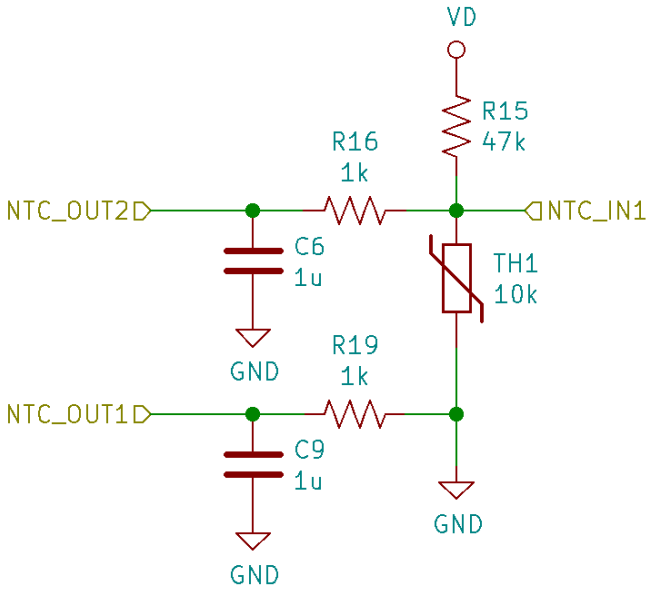

Figure 3.8: circuit for pseudo-differential NTC measurement

3.2.5 Power Supply

The IC is able to be powered by the monitored cells. By driving an external NPN transistor, the linear converter can provide a fixed voltage of 5.3V on the V_P pin (circuit in 3.9). The maximum current request of the device is stated as $I_{MAX} = 8mA$ when active and communicating. Also, V_P is used as supply for the NTC circuit, drawing $\sim 0.1mA$ for each NTC in the worst case, so additional 1mA should be considered for worst case calculations. When working at maximum cell voltage, that is $V_{max} = 16 * 4.2V = 67.2V$, the power dissipation of the NPN will be :

$$P_{MAX} = 8mA * 67.2V = 0.6mW$$

The transistor should also withstand $V_{CE} \geq 100V$ and have have an $h_{fe} \geq 100$. One recommended choice from the manufacturer is *ZXTN4004K* [8]. Resistor R18 in series with the collector limits current in the case of fault events and and shift a fraction of the dissipated power away from the transistor. It also provides filtering together with C7. The maximum value is given by formula 3.2

$$R_{MAX} = \frac{V_{min} - V_{CE,sat} - V_{P,max}}{I_{MAX}} \approx 900\Omega \tag{3.2}$$

- V_{min} is the minimum expected voltage of the cell string: 6*2V = 12V;
- $V_{CE,sat} = 0.25$ maximum, from transistor datasheet;
- $V_{P,max} = 5.5V$ from datasheet;

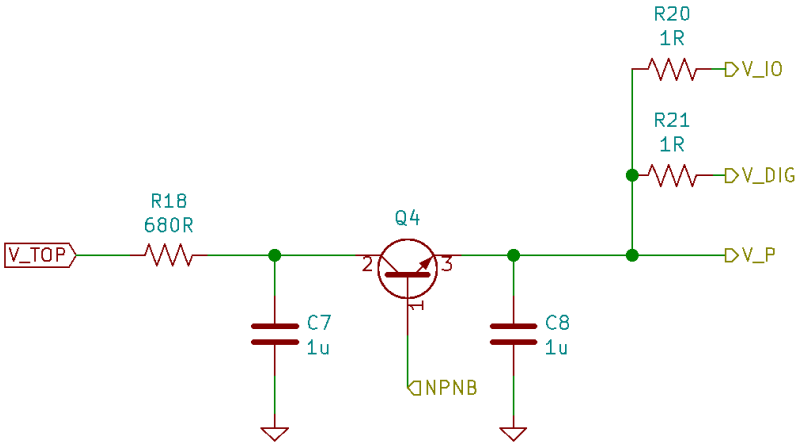

Figure 3.9: power supply circuit

V_{DIG} pin is used for the communication among *Slaves* in the chain and has 1Ω resistor in series as recommended, because it helps the loop stability of the converter. The same holds for V_{IO}, that is used for digital I/O and/or communication with the host via UART. In the case of the first device in the chain, V_{IO} must be externally supplied by the microcontroller in order to match voltage levels on communication

channels, so R20 can be left unmounted, while V_{IO} will be provided through an isolated converter [3.10]. This is to ensure insulation between the *Master*, which is a class A circuit, and the *Slaves* which are class B.

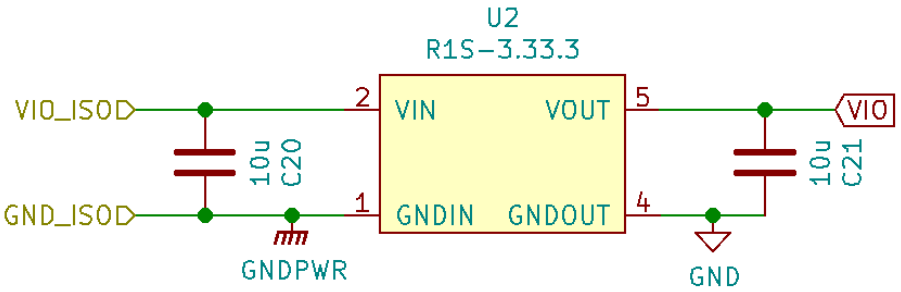

Figure 3.10: Isolated dc-dc converter

3.2.6 Chain communication

The communication is differential and capacitor isolated. When a message is sent from the host to the first device, it then transmits it up to the next device. The same is done when a device needs to send a message to the host. The speed is fixed to 4 Mbit/s and the protocol, even if not disclosed, is similar to an UART having 1 start bit, 1 byte data and 1 stop bit as stated by manufacturer. The communication has two buses, one used to pass data and commands, another to pass fault information regarding monitored parameters [3.11]. The data bus is bidirectional, with one HIGH/LOW bus to connect with higher/lower device in the stack. The fault bus instead is not used to pass data, but rather signal the presence of a fault to the lower devices by sending an "heartbeat", i.e. a pulsating voltage that, when absent, states a fault presence. The communication hardware can be seen in 3.11. It comprises of a series resistor, used to limit inrush current (10Ωon each end), a Transient Voltage Suppressor (TVS) diode used to absorb high voltage transients, and optionally one or two common-mode filters. These filters value depends on surrounding environment noise, so that it is recommended to use normally a single 51μH filter, or in strongly noisy environments two filters of 470μH and 100μH. It is also specified that total capacitance on each line should be less than 140pF to support all 16 stacked devices. Accounting for the capacitance of the common-mode filters and TVS diode (usually

one order of magnitude less) we can calculate in 3.3 the maximum cable length that can be used to connect two devices so that the total capacitance stays less than $140pF$, where C_{cable} is the capacitance per meter of the cable.

$$l_{cable} = \frac{140pF - 2C_{filter} - C_{TVS}}{C_{cable}} \tag{3.3}$$

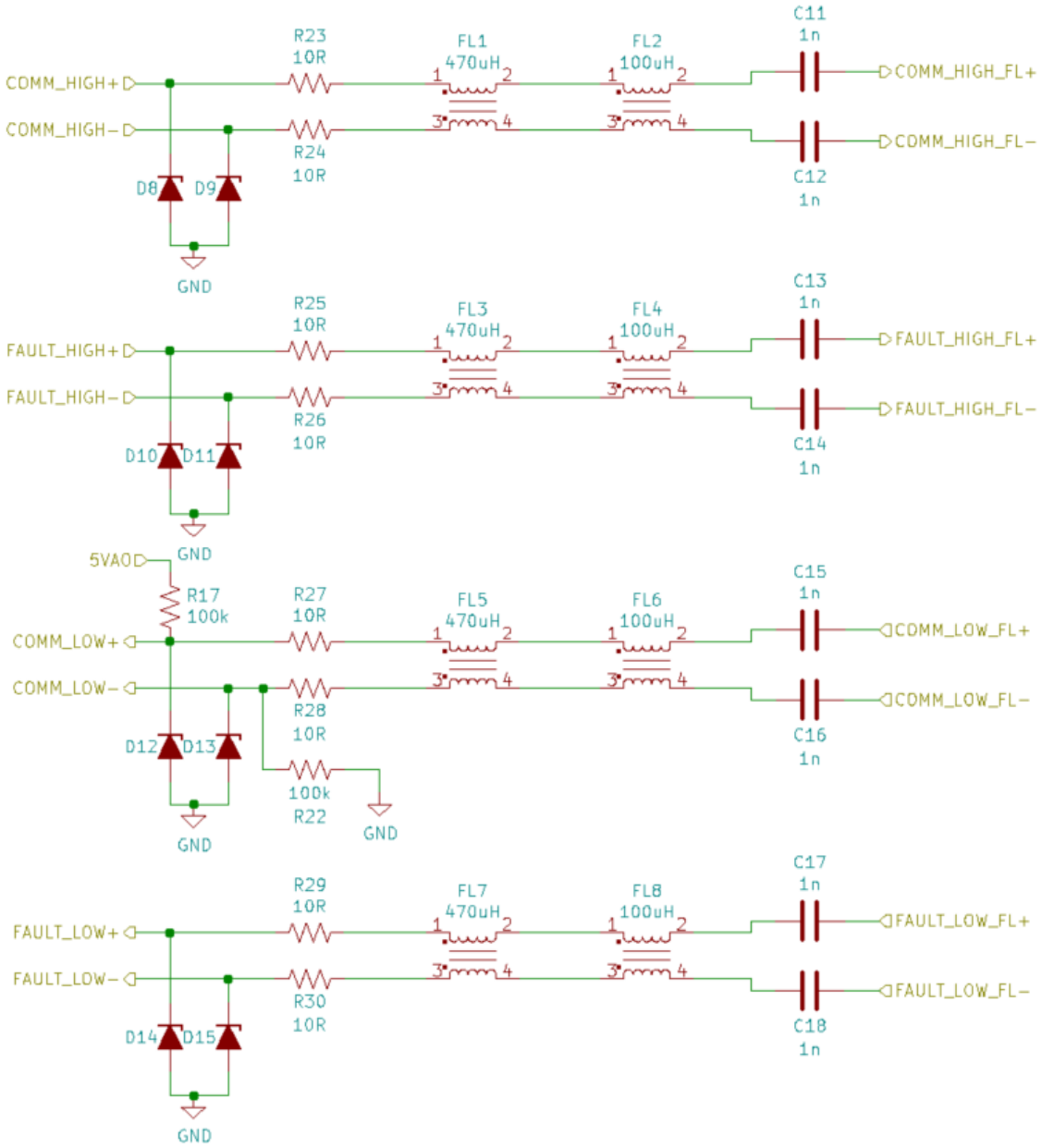

Figure 3.11: chain communication interface

Due to the fact that the first device in the chain has no lower device to be

connected to, the *COMM_LOW* bus can't be left unconnected, so a pull-up resistor to a 5V pin is placed on the + line and a pull-down to GND on the - line, both 100kΩ.

3.2.7 Host communication

The communication with the host takes place only with the first device in the chain via UART, but the protocol will be described in chapter 4. The host interface comprises also of a *FAULT_N* pin, which translates the described fault differential bus to a single active low signal, and a *WAKEUP* pin. This pin is used to set the device into SHUTDOWN mode, allowing for low power consumption when the battery is not used for long periods of time. The transition from ACTIVE to SHUTDOWN occurs if the *WAKEUP* pin is held low and a certain command via UART is sent from the host. The idea is to hold all the devices *WAKEUP* pin low with a pull down resistor, while connecting the first device pin to the host, then it will be possible to put all the devices in SHUTDOWN mode just by broadcasting the command. In order to pass from SHUTDOWN to ACTIVE it will be necessary to pull up the first device *WAKEUP* pin, then this will send a "wake tone" on the COMM_HIGH bus, waking the next device, and so on. The pin must be held high for enough time in order to wake up the first device, then the in order to start to communicate again with the chain, wake up time for the other devices must be accounted for.

Every communication line should be isolated for the reasons previously described. A single *IC* can be used like the *ISO7742* from Texas Instruments [9]. It provides 4 isolated buffer channels, 2 for each direction (schematic in 3.13) which ca be used as shown in 3.12 to isolate both the UART lines and the two digital I/O. The two enable signals (*EN1, EN2*) can put the respective outputs into high-impedance state if tied to ground, but in this case they are left open, that is output enabled. The isolator shall be mounted only for the first *Slave*. The pull-up resistor on RX line prevents the false triggering of a UART communication on higher devices in the stack.

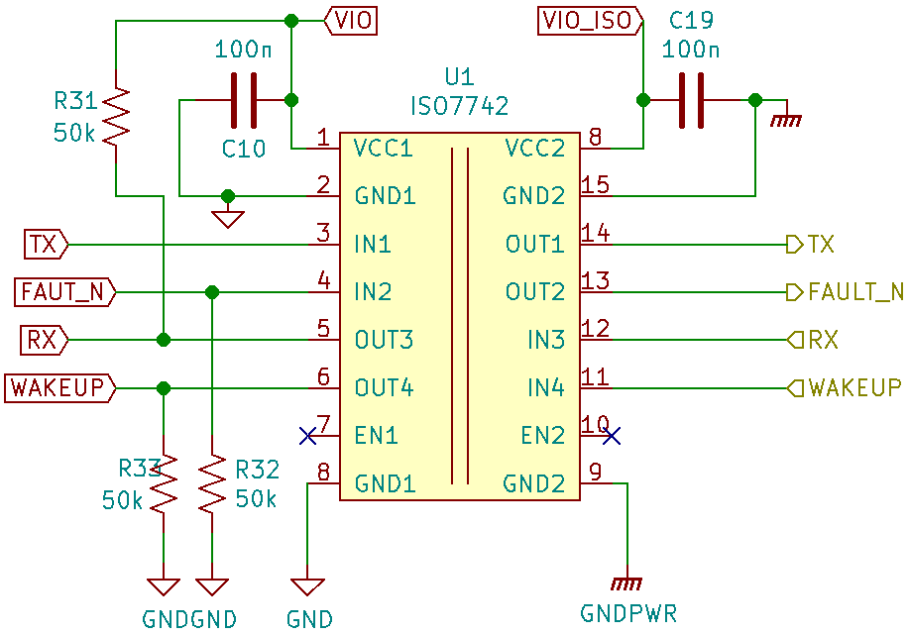

Figure 3.12: host communication interface

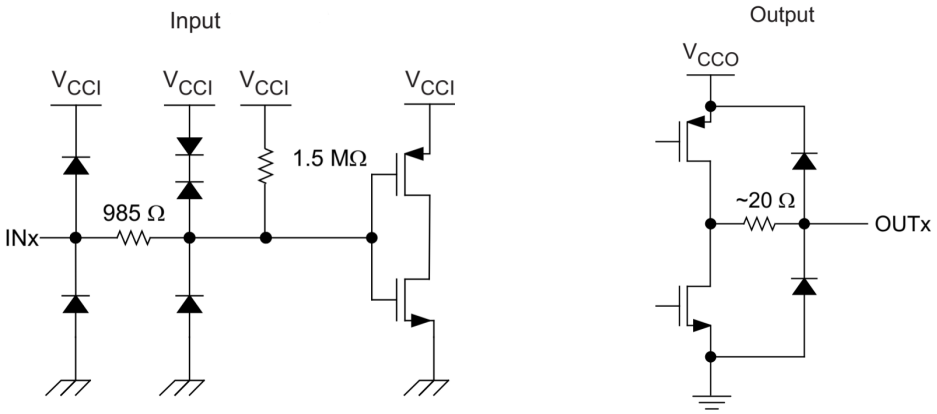

Figure 3.13: input and output circuit of the *ISO7742*

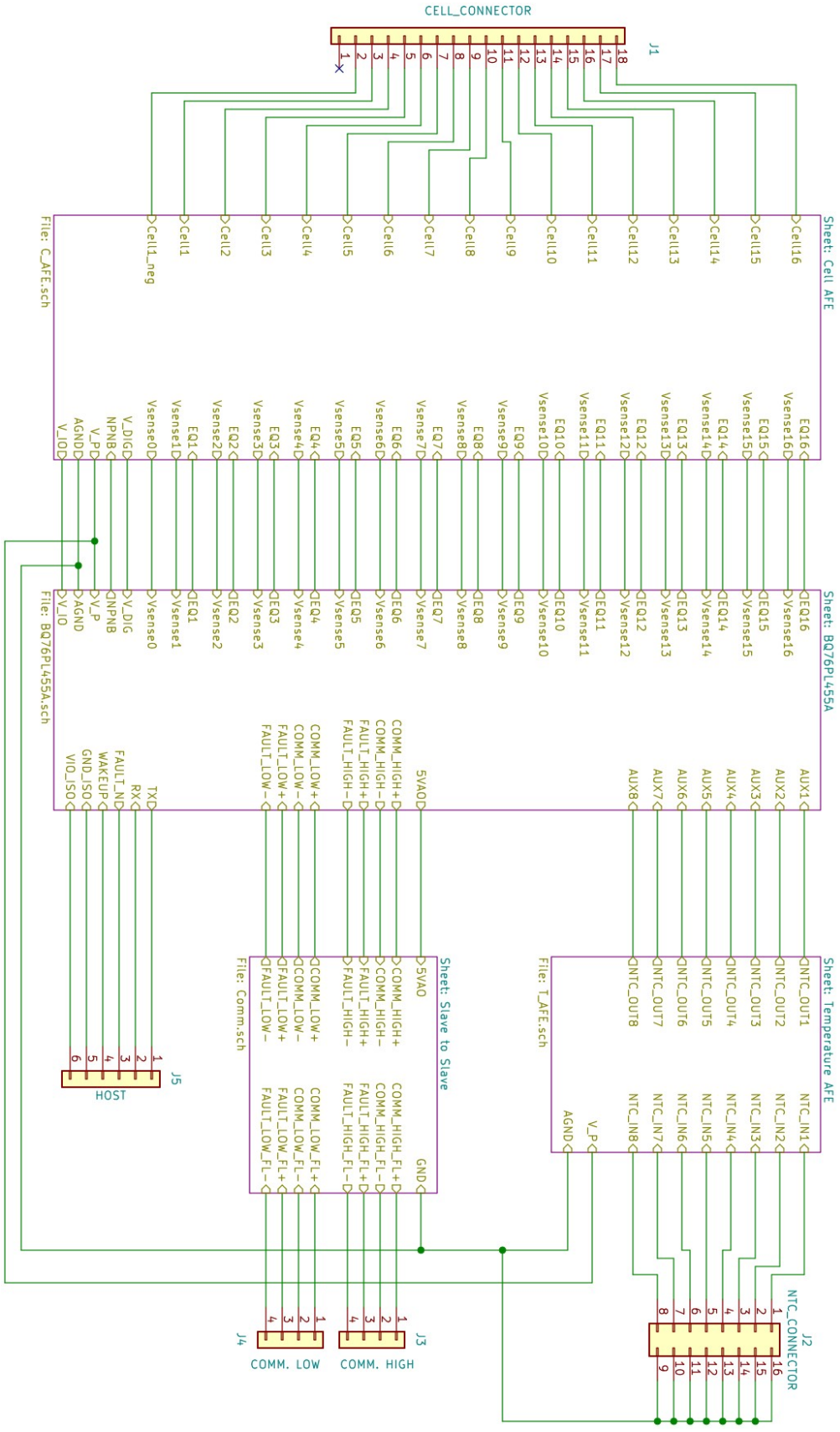

Figure 3.14: Complete *Slave* device schematic

Chapter 4

Master unit modeling

The *Master* module has multiple tasks to fulfill. As seen before, one of the task is to communicate with the *Slave* chain, but many other parameters need to be monitored inside a battery pack. Collected data need to be processed and sent to external controlling units of the vehicle, such as motor controller(s), Human-Machine Interface (HMI), Vehicle Management Unit (VMU) or others. Every aspect will be discussed in detail, in order to come up with a set of requirements for the *Master* unit.

4.1 HVIL and Contactors management

In a battery pack, the flow of current from the battery must be controlled and interrupted. High voltage must not be always present, i.e. when the vehicle is not used, and must be interrupted when hazards may arise or battery presents issues. For this purpose are used heavy load current relays, also called contactors, because of their ability to break circuits at high currents for a large amount of times and arc suppression. As a reference, in [10] the datasheet of a contactor from Tyco Electronics, with a current capability of 500A and an interruption capability of up to 2000A. The contactors are, in almost all cases, normally open (often referred as Form A) and are closed when energy flows inside the coil operating the switch. In this case the coil absorbs around 14W of power at 24V. The typical configuration (4.1) consists in one contactor on each battery pole. The low sides will be controlled directly by the *Master* unit, while the high side connection comes from a circuit called High Voltage Interlock (HVIL). The HVIL consists in a sequence of switches or physical disconnections that can interrupt the power supply of the contactors. For example, this power supply line is closed first on the ignition key, then passes on all high voltage connectors, so that in the event of mechanical failure the power is interrupted. Any other failure detection mechanism can act on the HVIL, such as crash detection sensors. The HVIL must not be confused with the High Voltage Disconnect (HVD), that physically breaks one or both cables of the high voltage bus, in fact HVIL is supplied by an external low voltage power source, usually 12 or 24V.

Although it is not technically a contactor, a precharge relay is placed in parallel to the positive contactor. Its purpose is to close the high voltage bus on a precharge resistor that will charge the big capacitors placed at inverters input, by limiting the current that would otherwise be uncontrolled and damage both capacitors and the battery. This relay can be of another type because of its different use with respect to the main contactors, e.g. a solid state one, with a much smaller current and power switching capability.

The interface for controlling the three relays is shown in 4.2 in the simplest of the

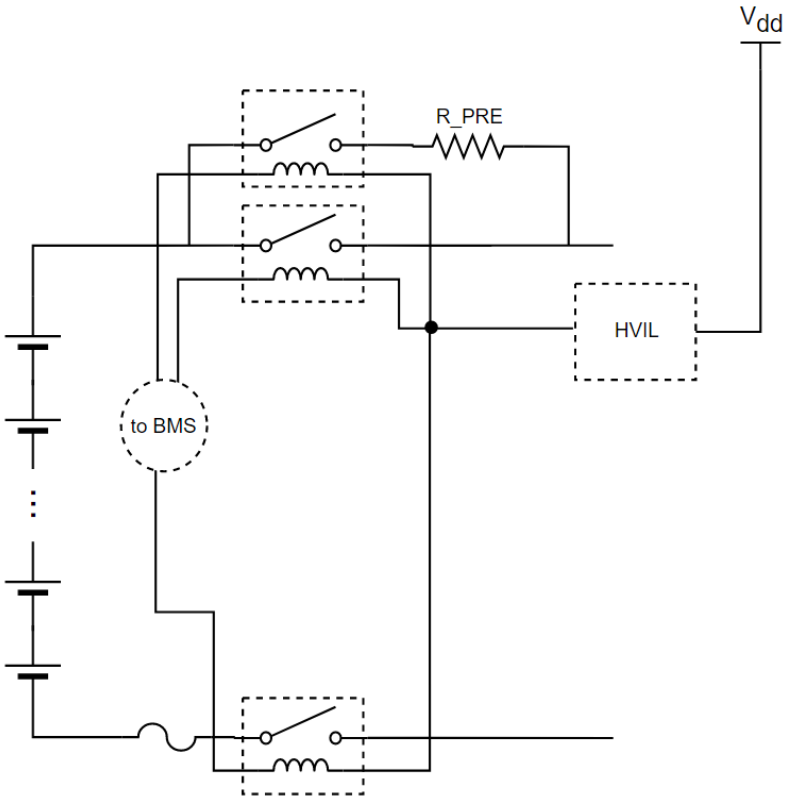

Figure 4.1: Typical structure of battery safety components

forms. One NMOS is used for pulling down each coil of the relays. Since the coil is an inductive load, when the current is interrupted, either because the HVIL has interrupted the high side or we just want to open the relay, it will try to keep the current constat by creating a reverse voltage that can destroy the mosfet or the switch that opened the HVIL. A free wheeling diode between the low side and the high side of the coil can prevent these effects, as the current will flow into the diode wh. The side effect is that the relay will take more time to open, and could be a problem in high speed switching applications, but this is not the case. Diode must be chosen according to its maximum reverse voltage (i.e. the contactors supply) and the current flowing into the coil.

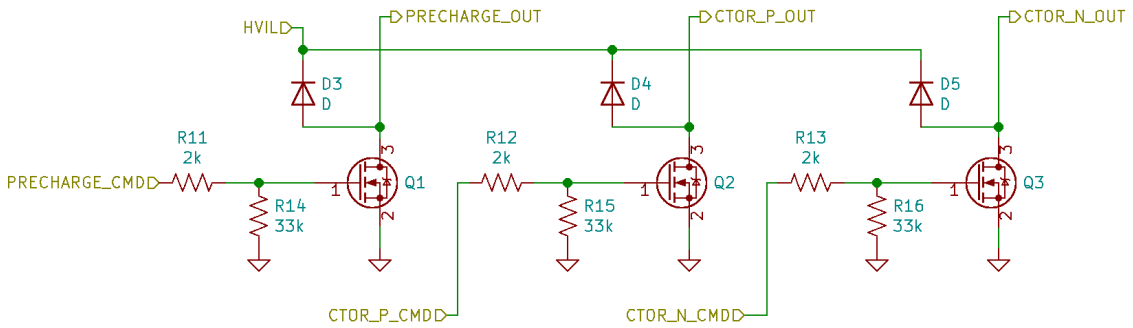

Figure 4.2: Hardware for controlling the relays

It could be useful to monitor the HVIL so that start-up sequence can be prevented if HVIL is already missing or if there is an error while contactors are closed. The circuit in 4.3 scales down the HVIL voltage by ten with a resistor divider (R17, R19), for a maximum of 33V input. A rail to rail Operational Amplifier (OpAmp) is used in buffer configuration with a low pass filter in output (R18, C12). The input is protected from overvoltage and undervoltage by two diodes (D7) which are commonly found in a single package for this purpose.

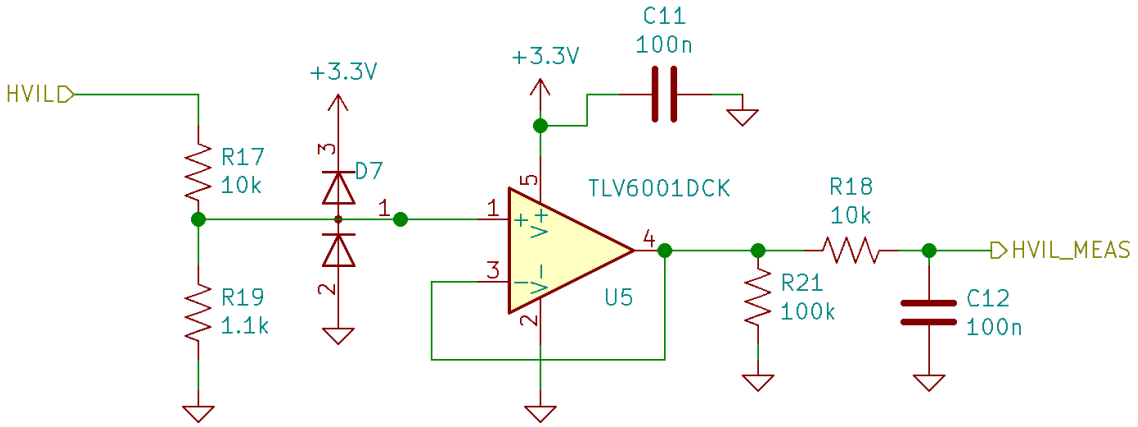

Figure 4.3: Measuring HVIL voltage

4.2 High Voltage measurement

In order to detect correct operating condition, the voltage on the bus must be measured at least in two points, shown in 4.4, that is before and after the contactors, often referred as V_{BATT} and V_{BUS}. Any discrepancy between the intended operational state, i.e. relay open/closed, and the measured voltage in the two points will result in an error detection, for example:

- $V_{BATT} = 0V$: fuse blown or damaged connection between cells or disconnected cable
- $V_{BATT} > V_{min} \& V_{BUS} = 0V$ & Contactor Command = closed : one or both contactors are stuck open
- $V_{BATT} = V_{BUS} > V_{min}$ & Contactor Command = open: both contactors are stuck closed

The list is not exhaustive and obviously depends on the overall architecture of the vehicle. More measured points would of course allow to cover more error cases and identify precisely the error source.

In order to measure an high voltage circuit while maintaining isolation, there is basically one way to achieve it, that is to locally convert the analog signal with an A2D circuit and send the digital signal through an insulating barrier, for example with an optocoupler. There are many market solutions that integrate these functions in to single packages, in this case the *AMC1311* from Texas Instruments. Its schematic can be seen in 4.5, and it consists in a $\Delta\Sigma$ modulator with a single ended input that converts the signal into a digital bitstream sent through a SiO_2 barrier, converting again the signal into an analog one through a filter, with a gain of 1 [11]. The adopted implementation scheme is in 4.7.

The voltage is measured through a resistor divider (R4, R5, R9) so that the input V_{IN} does not exceed its limit, being the full scale range 2V. For $V_{BATT,max} = 800V$, R4=R5=4.22MΩare used, while R9=21kΩ. This allows to use the entire full scale range: $V_{IN} = 1.985V@800V$. Surface mount resistors can't withstand such high

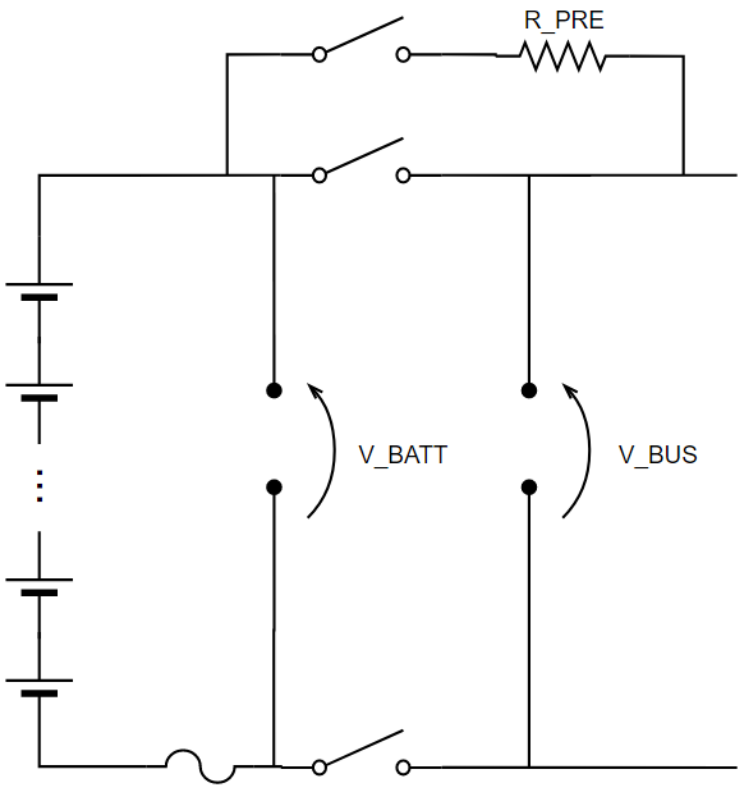

Figure 4.4: High voltage bus measurement points

voltage ($\sim 400V$) so through-hole axial resistors must be used, whose lowest tolerance value is however 1%. R9 can be chosen as surface mount type with 0.5% tolerance as it is a commonly available product. The output is converted in single ended with the use of a rail to rail OpAmp, U4, with a gain of 1 and an offset of 0.5V through U3, so that the output ranges from 0.5 to 2.5V. The differential stage has a low pass filter, through C10 and C6, with $f_{cut} = 16kHz$.
The high voltage domain of the IC must be powered from a 5V source. This can be obtained as in section 3.2.5 with an integrated monolithic isolated DC-DC converter or with discrete components, that although occupying more area it is cheaper. The regulator consists in a specialized push-pull driver, *SN6501* [12], driving a transformer in push-pull configuration [4.6].

This allows a flexible circuit that can be tuned by choosing the transformer to achieve wanted output voltage and isolation. The driver can work with 3.3V or 5V supply, in this case a 3.3V supply will be considered for reasons that will be seen later. The transformer chosen is from Wurth Electronics (part number 760390015)

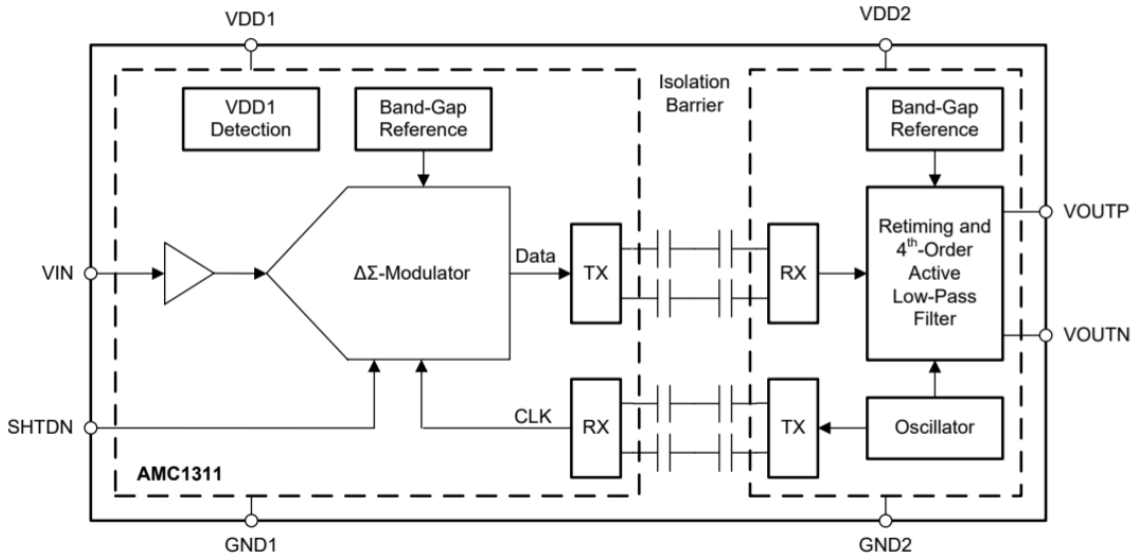

Figure 4.5: AMC1311 overview

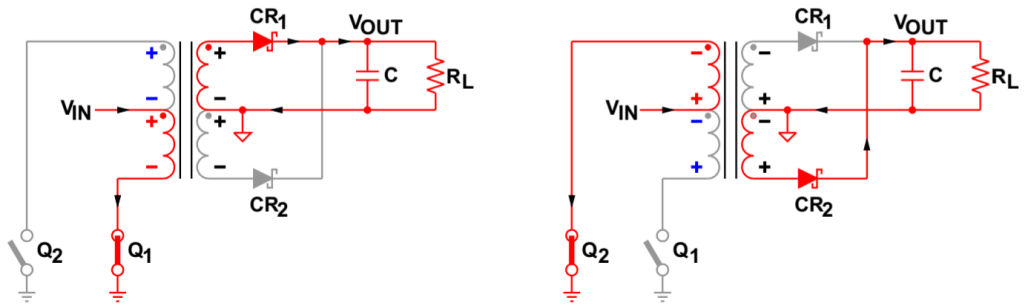

Figure 4.6: The two phases of a Push-Pull converter

and has a turn ratio of $1 : 2.1 \pm 3\%$ with 2.5kV isolation. The schottky diodes on secondary side are MBR0520L from Fairchild Semiconductors, because of their low V_f and $V_{r,max} = 20V$. The output is unregulated and could be used for wide input voltage applications, or can be further regulated with an LDO converter, as in this case. Full schematic in 4.7. It is important to notice that, in order to actually measure the two voltages shown in 4.4, two circuit in 4.7 needs to be replicated exactly as it is, by changing the high voltage inputs. In this way each circuit can measure the voltage independently when the contactors are open, while when they are closed the two isolated power supply ciruits will be in parallel, and thus redundant.

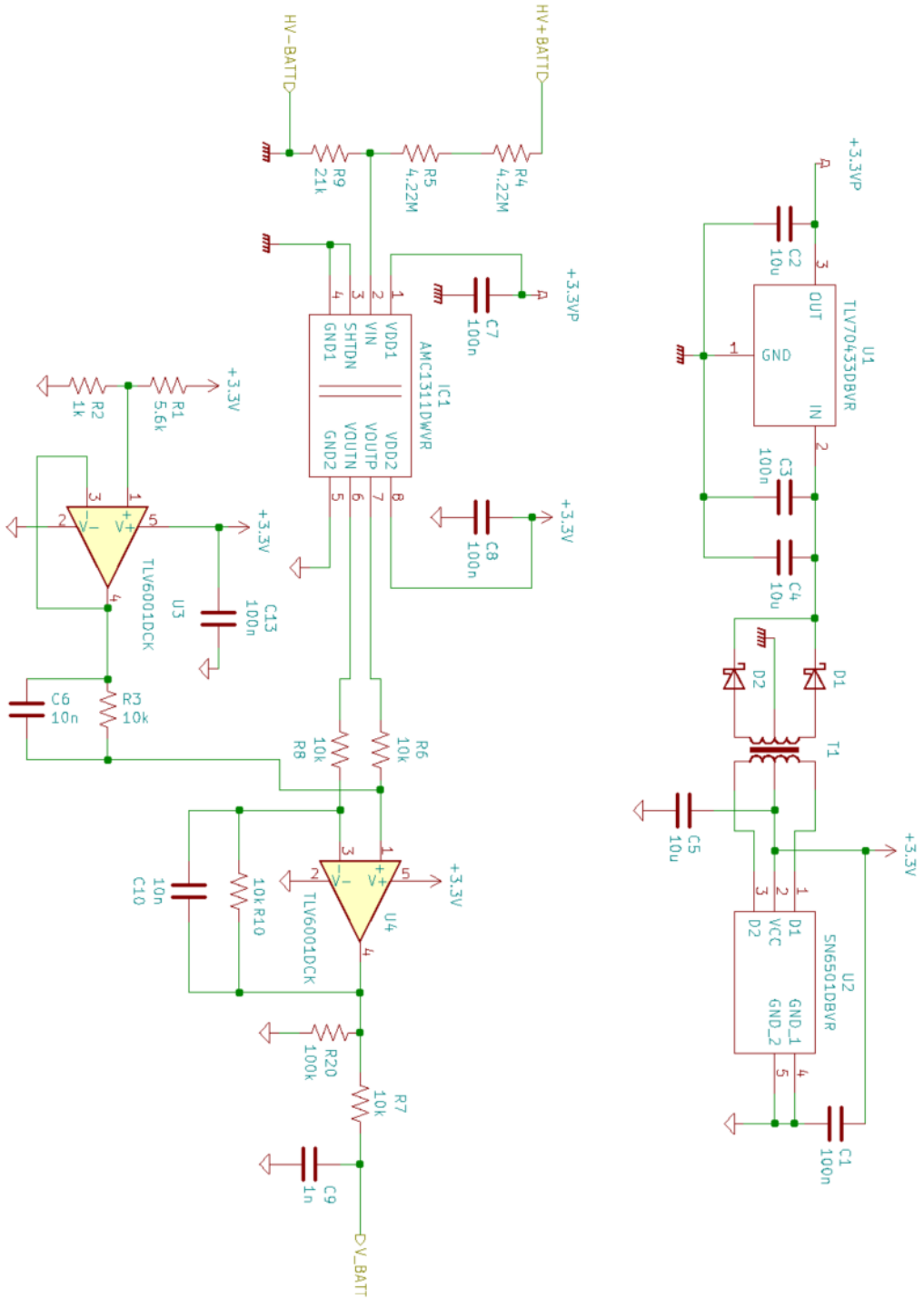

Figure 4.7: High voltage measurement circuit

4.3 Current sensing

Measuring the current flowing in and out the battery pack is of fundamental importance, because:

- maximum charge/discharge current indicated by cell manufacturer need to be observed
- is used to calculate energy used/stored for SoC calculation
- observing current levels throughout the battery life determines SoH

two are the most common ways to obtain a measure of the current, and they will be both discussed to point out strength and weakness of these methods.

4.3.1 Shunt based method

By placing a resistor in series with one terminal of the battery it is possible to measure a differential voltage across it when current is flowing in both direction. The resulting measurement has an excellent linearity and minimal offset error due to today high precision resistors manufacturing, but of course a certain amount of power will be dissipated on the resistor. The choice of resistor value depends on the battery maximum current output. Based on some products available on the mainstream market, the range of possible measurements in shown in 4.1.

P_{max}	R	I_{max}	V_{FSR}
36W	$100\mu\Omega$	$\pm 600A$	$\pm 60mV$
	$250\mu\Omega$	$\pm 380A$	$\pm 90mV$
25W	$500\mu\Omega$	$\pm 220A$	$\pm 110mV$

Table 4.1

Since the resistor is placed on a "hot side" (i.e. on high voltage bus), a solution similar to what described in section 4.2 can be adopted, by using a specialized isolated differential amplifier similar to the *AMC1311*, called *AMC1301*, that has a $\pm 250mVV_{FSR}$ and a gain of 8.2 on the (differential) output. This solution can be much cheaper, at the same performance, if compared to the next solution.

4.3.2 Hall effect-based method

By exploiting Faraday's Law of current it is possible to measure the magnetic flux generated in a magnetic core when a current flows in a conductor passing through it. This technology uses a flux gate element, that can be used in a open or closed loop configuration [4.8].

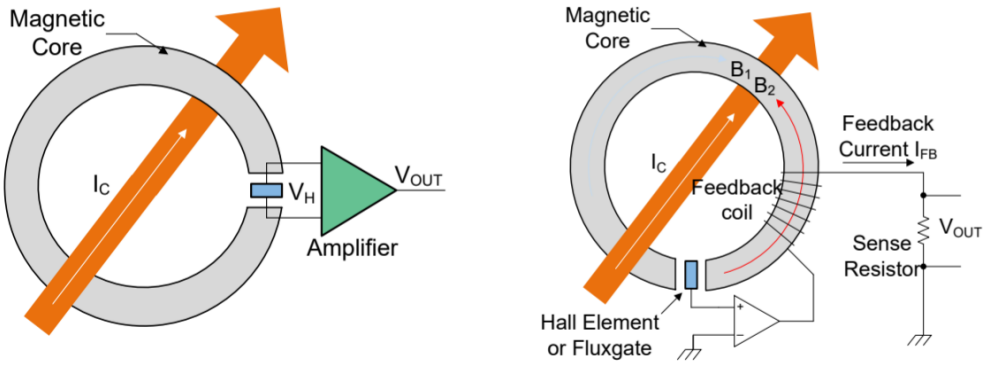

Figure 4.8: Hall effect current sensors: Open (left) and Closed (right) loop [1]

An open loop sensor has a built-in amplifier circuit so that the output can be input to an ADC without much further components needed, while the closed loop sensor has a current output which need to be sensed via a sense resistor and a differential amplifier. The advantages of (both) hall effect current sensor over shunt resistor method are:

- Built-in voltage isolation, output on the "cold side" can be sensed without the need of ensuring isolation
- Power is supplied by a low voltage source, power consumption two or more order of magnitude lower

The disadvantage are cost and worse accuracy overall, which can range from 0.5% to 3.5% in general, with an exponential cost increase for higher accuracy devices. In 4.9 it can be seen the accuracy over temperature of an automotive open loop sensor [13] with two output channels optimized for resolution, with $\pm 100A$ and $\pm 350A$ range respectively.

Parameter	Symbol	Unit	Temperature					
			−40 °C	−20 °C	0 °C	25 °C	65 °C	125 °C
Accuracy @ 0 A	X	A	0.35	0.29	0.23	0.15	0.23	0.35
Accuracy @ ±50 A			1.50	1.25	1.01	0.70	1.02	1.50
Accuracy @ ±100 A			3.75	2.98	2.21	1.25	2.25	3.75

Parameter	Symbol	Unit	Temperature					
			−40 °C	−20 °C	0 °C	25 °C	65 °C	125 °C
Accuracy @ 0 A	X	A	1.50	1.35	1.19	1.00	1.20	1.50
Accuracy @ ±175 A			5.25	4.71	4.17	3.50	4.20	5.25
Accuracy @ ±350 A			9.25	7.94	6.63	5.00	6.70	9.25

Figure 4.9: Accuracy data of low (top) and high (bottom) current channels

The output has a bias of 2.5V at 0V and a gain of 20mV/A and 5.7mV/A for the two channels, for a total of 4V swing. The resolution is however the same, 2.5mV, leading to a resolution of 55mA and 440mA respectively. The bandwidth is limited to 1.1kHz On the other hand is easy to provide an hardware interface for reading such sensors, as shown in 4.10.

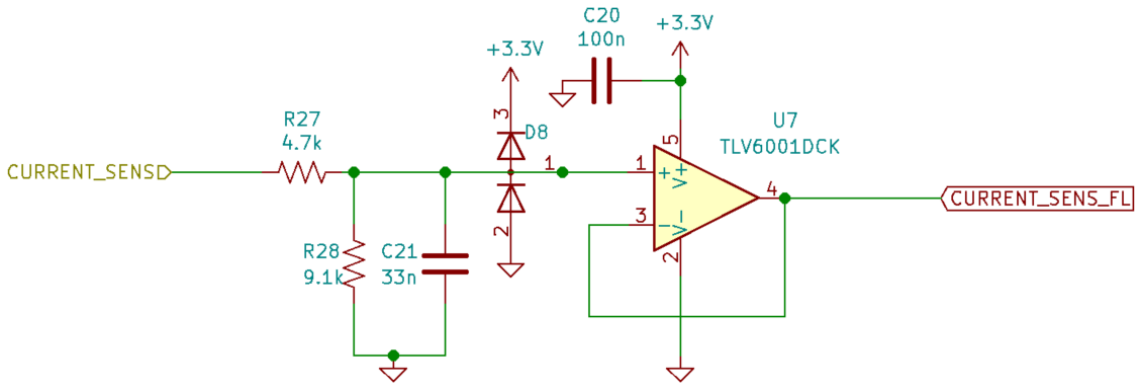

Figure 4.10: Current sensor input

The voltage is scaled from 5V to 3.3V with a voltage divider at the input of a rail to rail OpAmp. Capacitor C21 creates a low pass filter with $f_{cut} = 1.5kHz$. The values chosen depends on the sensor, in order to respect maximum load ratings of current and capacitance.

4.4 Isolation resistance monitoring

In order to ensure protection against electrical shock and hazards, the high voltage bus must maintain a certain isolation with the GLVS (Grounded Low Voltage System), i.e. the chassis and any conductive part of the vehicle. The isolation is measured by the ohmic resistance between them, and the minimum value depends on the maximum voltage of the bus and whether is AC or DC. [5] states the minimum resistance value as 100Ω/V for DC and 500Ω/V for AC, with some exceptions. Usually the measurement takes place in more than one physical part of the Grounded Low Voltage System, for example inside the battery pack grounded case and in another part of the chassis of the vehicle. I will not cover the methods to measure the isolation resistance as this function is often left to stand-alone devices. However the BMS must be aware of this parameter and monitor it for keeping the battery in safe operating conditions, thus must show an adequate interface for reading an Isolation Monitoring Device (IMD). For instance, an automotive IMD from Bender [14] is able to measure isolation resistance with respect to two chassis points and send the result through two signals: a Pulse Width Modulated (PWM) signal with changing frequency (10Hz to 50Hz) carrying the resistance value and a secondary parameter ,and a digital signal indicating if the resistance is above/below a preset threshold. Both signals are open emitter circuits, switching to $KL15-2V$ according to datasheet. The circuit in 4.11 can handle both signals offering a robust protection

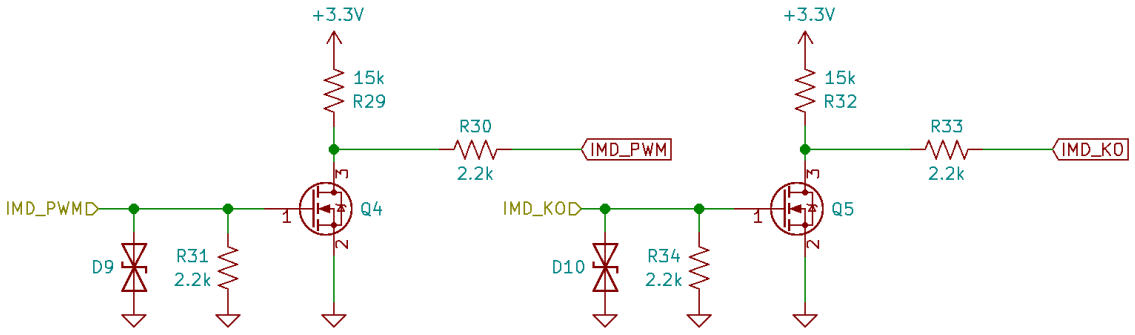

Figure 4.11: IMD signals input

against ESD and EMC, and by choosing a MOSFET with an adequate $V_{GS,max}$ can

work with a wide range of supply voltages (notice that it is an inverting configuration). The digital I/O is also fail safe in case of disconnection, since a digital 0 correspond to a fault.

4.5 CAN bus communication

The most used communication method as of today automotive standards is the Controller Area Network (CAN). It is a multi-master bus which has good noise immunity and allows for a large number of node connected, consisting of only a twisted-pair cable (differential communication). Some concept of the bus will be presented in order to understand this section, but a detailed description will not given and is left to the reader. The CAN protocol is defined in the ISO-11898 as CSMA/CD+AMP, that is:

- **C**arrier-**S**ense **M**ultiple-**A**ccess: every node can access the bus, waiting for a certain amount of time before trying to send a message
- **C**ollision **D**etection and **A**rbitration on **M**essage **P**riority: each node can detect a collision between the actual bus state and the attempted transmission, by a predefined message priority.

The protocol consists of frames sent by the nodes, shown in (4.12), that can contain at most 8 data bytes. The identifier is 11 bits in the *Standard* CAN and 29 bits in the *Extended* CAN. Maximum baud rate is 1 Mbps.

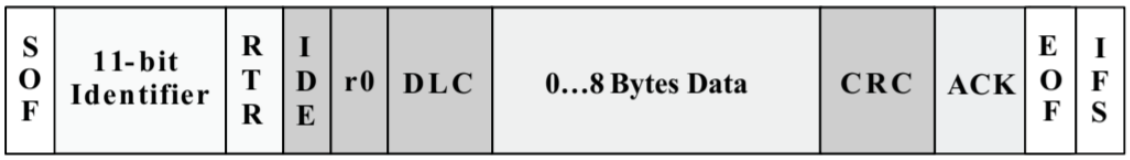

Figure 4.12: Standard CAN message frame [2]

A line driver is needed to interface the bus in order to ensure correct arbitration: In fact these drivers implement the hardware layer required by the ISO standard and sit between the bus and the CAN controller implemented by the node. The bus requires two line termination of 120Ωat each (physical) end 4.13.

A new standard, called CAN Flexible Data rate (FD), can be used to increase the baud rate during data transmission portion of the frame while increasing the number of possible data bytes transfer in a single frame up to 64, allowing a huge

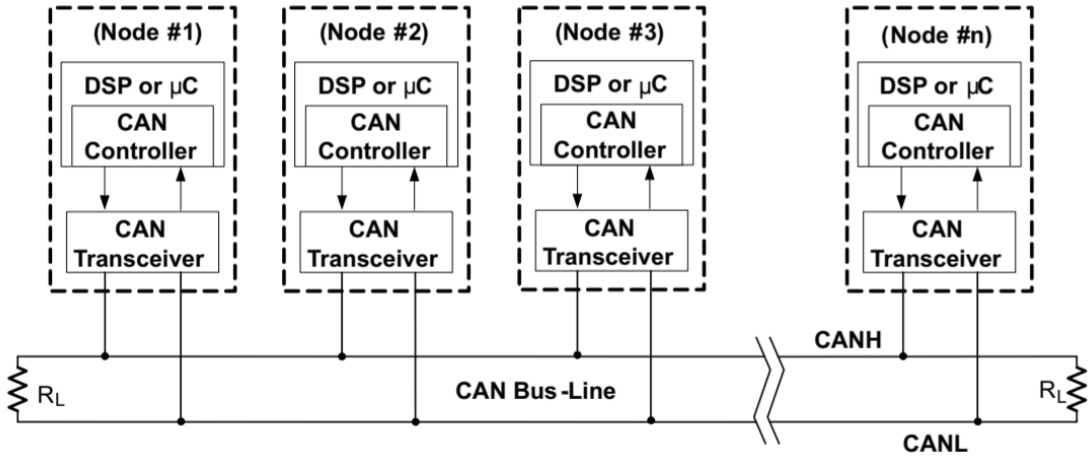

Figure 4.13: CAN bus physical layer [2]

increase in throughput. The physical layer of the CAN FD bus is compatible with the standard CAN bus. Due to the increasing need of data transmission speed, it is a wise choice to adopt a CAN FD capable node so that it will be possible to exploit this functionality if requested by the application.

The chosen transceiver is *TCAN1051V-Q1*, circuit in 4.14.

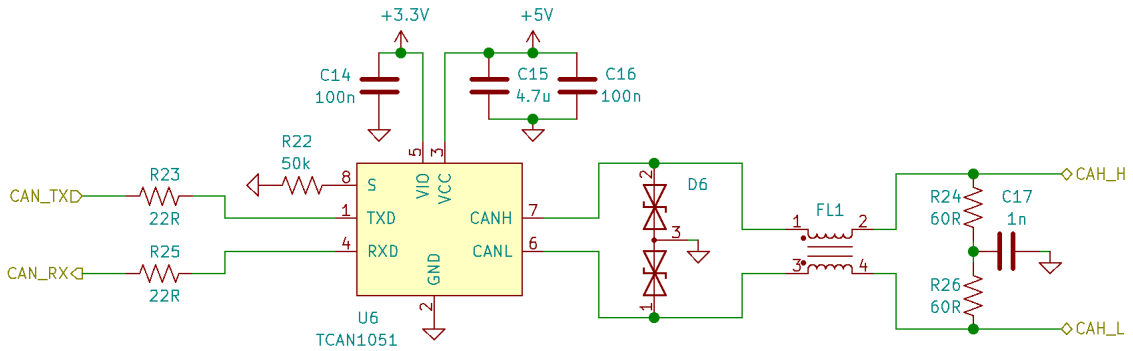

Figure 4.14: CAN transceiver circuit

The transceiver is able to transform a serial communication (TX and RX pins) in a differential communication for the CAN bus (4.15) and is capable of FD function. The drivers work at 5V supply while the serial communication has a dedicated supply pin should the working voltage differ (3.3V in this case). The select pin **S** can be pulled high if only "listening" is needed, so it is tied to ground in this case.

The transceiver pins are protected from ESD with TVS diodes(D6). The use of

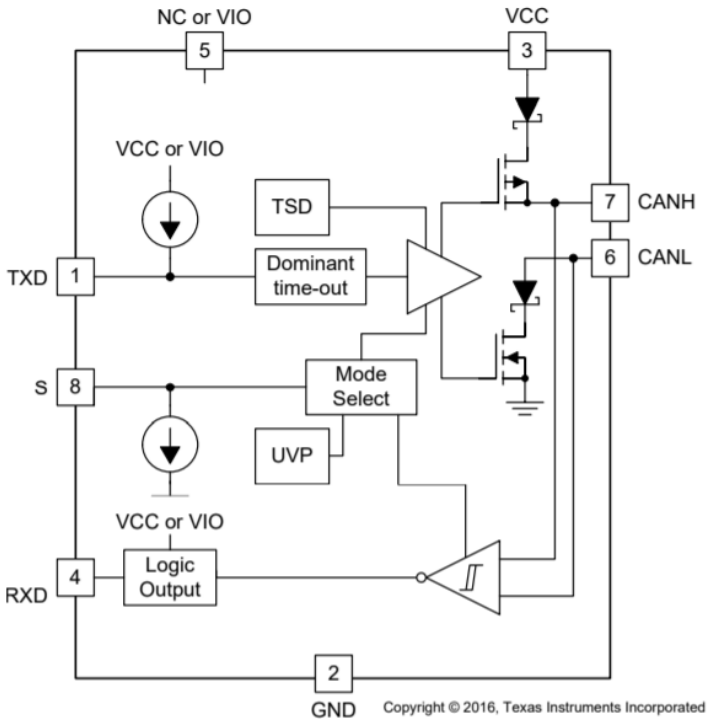

Figure 4.15: transceiver schematic

a common mode choke (FL1) can greatly increase system reliability for Electromagnetic Compatibility (EMC), filtering high frequency noises from the node to the bus and vice versa. But as pointed in [15], this could lead to transceiver damage due to high voltage transients if a short circuit of the bus to a dc voltage should occur, caused by the inductive flyback of the choke itself. TVS diodes placed before the choke and integrated protections inside the transceiver offer some immunity to these transients, but if the application allows it, it is advised not use them. However, in the same report, many automotive chokes have been tested, and results show that the ones based on a toroidal core generates lower transients, so they are preferable. Although it is not advised to place a termination on a node but rather on the bus lines, due to the fact that removing that node would effectively kill the bus, it is offered the option to mount the termination on the PCB. The splitting termination (R24, R26) creates a filter with C17 which helps coupling high frequency noises to a solid ground potential. The value of the capacitor depends on the speed of the bus, as

$$f_{cut} = \frac{1}{2\pi * 120\Omega * C}$$

For a 1 Mbps bus, a 4.7nF capacitor creates a 3dB point at 1.1 MHz. Instead for a CAN FD bus at, for instance, 5 Mbps during data transmission phase, a 1nF capacitor moves the 3dB point at 5.17 Mhz. The resistor needs to be matched and have an appropriate power rating in order to withstand a short circuit of the bus to dc voltages.

4.6 SoH and SoC

Determining the State-of-Charge and State-of-Health of batteries is a long discussed topic. The definition of SoC can be given as the percentage on the remaining capacity over the maximum capacity of a battery:

$$SoC = \frac{Q_{rem}}{Q_{max}} x100\% \tag{4.1}$$

The SoH can be defined as the actual maximum capacity of a battery over the nominal one:

$$SoH = \frac{Q_{max,act}}{Q_{max,nom}} x100\%$$

The health of a battery is an important parameter in a vehicle, as it estimates with more accuracy the possible driving range for the user, but also indicate when the battery need to be changed, with a threshold of about 80% in general. This parameter is mainly due to the aging of the battery electrochemical properties, which can vary depending on how the battery has been used (temperature and current conditions) over its life.
The simplest method to calculate SoC is by Coulomb Counting (CC) method:

$$SoC_t = SoC_{t_0} - \frac{1}{Q_{max}} \int_{t_0}^{t} \eta I_{batt} \tag{4.2}$$

the meaning is that the SoC value (at time t) is obtained from the initial SoC at start time (t_0) by subtracting the integral over this time of the current flowing from/in the battery. The parameter η accounts for the coulombic efficiency of discharge and charge processes. This method has low impact on processing power, but its inaccuracy is due to:

- knowledge of SoC_{t_0}: how to choose this initial parameter?
- accumulated error over the integral due to current sensor intrinsic error

The difficulty of determining the SoC of a battery reside in the impossibility to establish a one to one correlation between it and the voltage measured at the cell

terminal. In fact, battery voltage and its energy are related by multiple factors as temperature, charge/discharge rate and cycle time, as shown in figure 4.16 reported from [16].

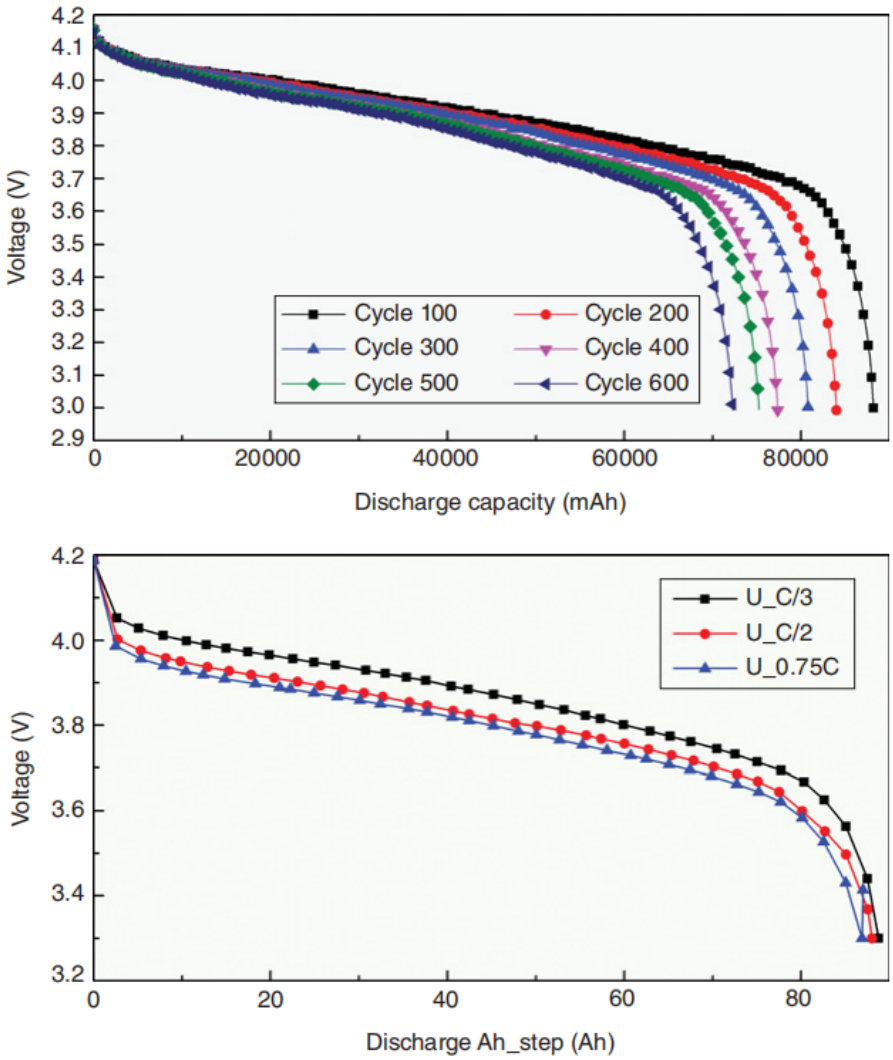

Figure 4.16: Capacity vs Voltage curves of a cell with a nominal capacity of 90Ah

There is instead a correspondence between energy and Open Circuit Voltage (OCV), i.e. the voltage at cell terminals when no load is applied. The OCV is obtained when the battery is at rest for a certain amount of time, due to an effect called voltage recovery 4.17.

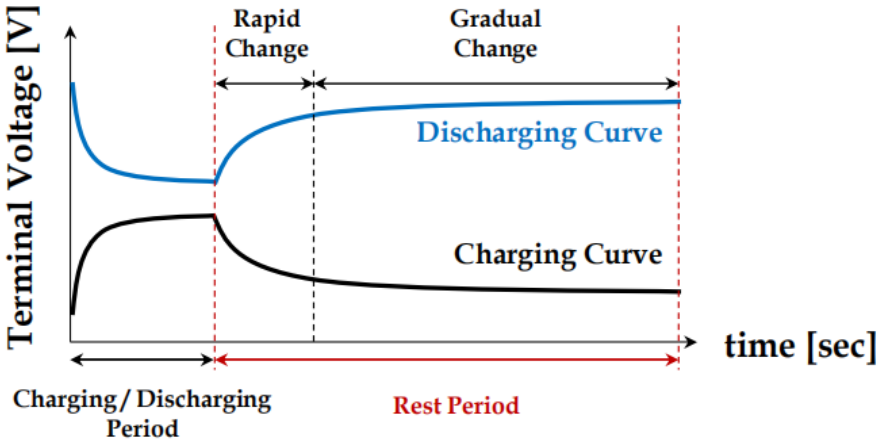

Figure 4.17: Voltage recovery effect after charge/discharge [3]

This is better understood if the equivalent circuit model of a cell is observed. Many of them have been formulated, but a compromise between simulation complexity and accuracy has been found to be the Thevenin circuit model [17] shown in 4.18. The parasitic resistance and capacitance R_p and C_p explain the voltage relaxation effect, while $R\Omega$ is the cell intrinsic resistance, that accounts for η, thus wasted heat.

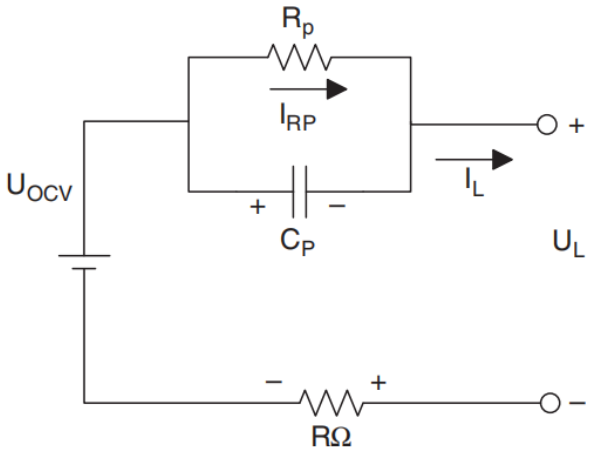

Figure 4.18: Thevenin equivalent circuit model

In order to accurately estimate the SoC, many studies in literature [17] [18] adopted equivalent circuit models like the previous and identified their parameters through recursive algorithm and Kalman Filters.

Since the scope of the work is not to provide an algorithm for SoC estimation, as this is a parallel subject heavily correlated to the used cells for the battery pack, but rather providing an hardware platform capable of running such algorithms, this topic will not be further detailed. Depending on the complexity and required speed of the chosen algorithm, it could be necessary to introduce a DSP or a FPGA in order to create specific hardware structures supporting them, without taxing the microcontroller, that would otherwise not be able to complete all the required work in time.

4.7 Additional considerations

In addition to what previously described, there could be some other features that need to be implemented by the *Master* unit that mainly depends on the level of integration of the BMS in the overall vehicle architecture, i.e. how information are passed between the BMS and others ECU.

Depending on the battery pack construction, it could be necessary for instance to monitor the pressure inside the pack for sealed case and/or humidity and water leakage for water cooled batteries. It can be wise to provide additional analog inputs with the same interface as seen in 4.10, that can handle typical 3-wire sensors (5V power supply and 0.5-4.5V analog output). Another option can be the use of digital integrated sensors relying on silicon sensing technologies, which can monitor more than one parameter and provide the result on digital communication buses. One example is the *HDC2021* [19] which uses capacitive based sensing for relative humidity and a bandgap reference for temperature sensor, sending information on a *I2C* bus. This solution can be less accurate, due to size, position and intrinsic accuracy of integrated miniature sensors, but strongly cost effective.

Based on these readings, the BMS can potentially be able to handle the cooling system of the battery by itself. This is useful when the BMS is developed as part of a power train system that will be used in many vehicles and sold to customers.

4.8 Microcontroller

The chosen microcontroller is *S32K144* from NXP. Its characteristics make it suitable for this job for a number of reasons.

- It is an automotive certified AEC-Q100 microcontroller, which is a mandatory requisite.
- Easy to implement as a prototype thanks to available evaluation boards and a dedicated IDE
- Already used inside the company, so faster development due to a deeper knowledge of the product (both HW and SW).
- A complete set of peripherals for the needed communication buses: 2 CAN bus ports (one with FD capability) and 3 each of UARTs, SPI, I2C.
- Has a Cortex-M4F core, which has a Single precision Floating Point Unit. Low power and High Speed up to 112MHz mode available.
- 2 ADCs with 12 bit resolution and 8 channels each, 32 timer channels (16 bit), DMA with 16 channels and 63 configurable trigger inputs

In the next section it will be described how the microcontroller peripherals will be programmed and exploited in order to implement all the functions discussed up to now. The full schematic for the *Master* unit is 4.19

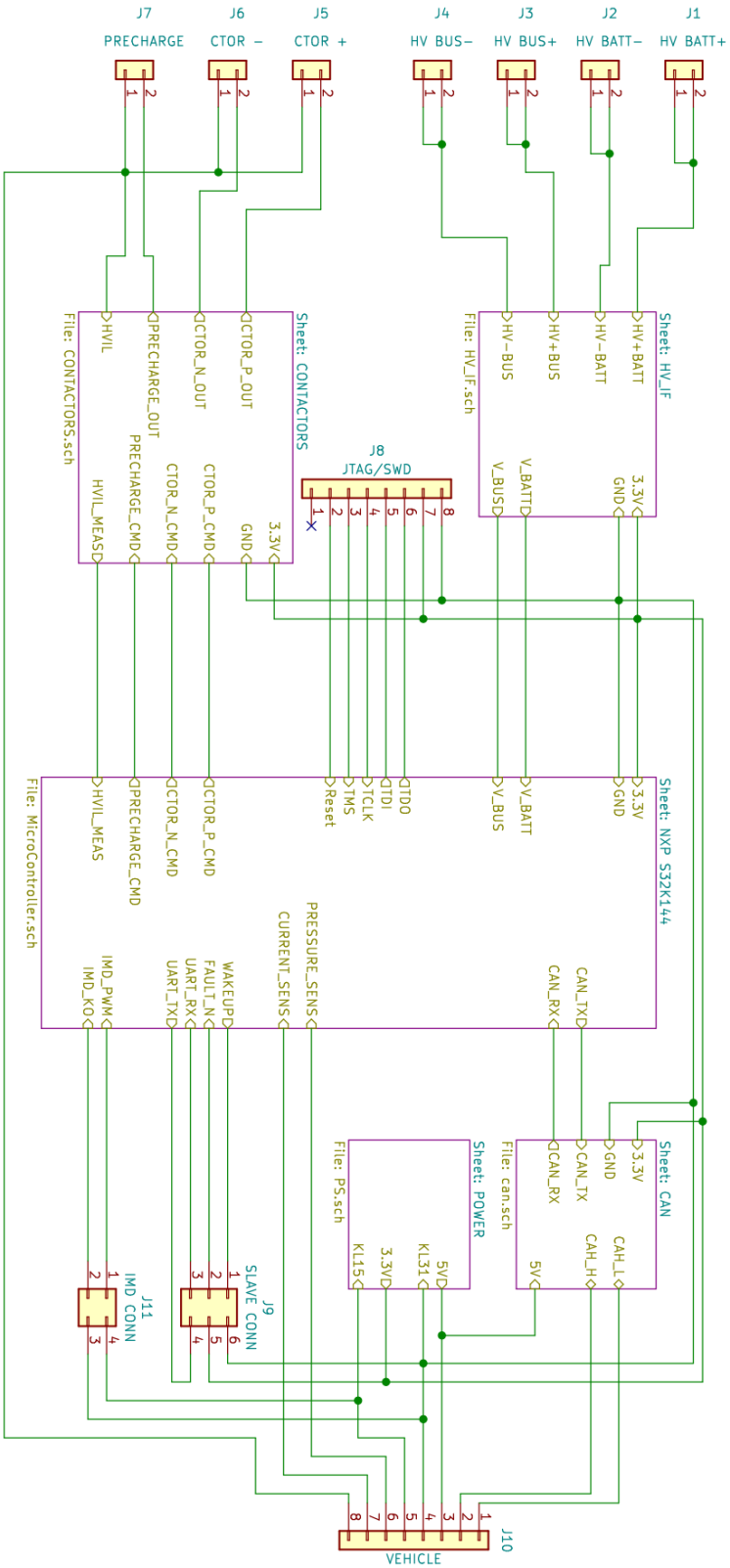

Figure 4.19: Master unit schematic

Chapter 5

Software implementation

5.1 General Structure

No Real Time Operative System has been adopted in the first place. The simple structure in 5.1 allows for a precise timed routine. The initialization phase runs once and configures all the required peripherals. The routine is timed through the use of a timer, which is configured to fire an interrupt at a fixed rate. The Interrupt Service Routine (ISR) triggered by this timer increments a free running counter, used to check whether the code can be executed or not. In the scheme there are three different pieces of code that run with three different frequencies, but they could be also just one or more than 3. In order to assure that every piece of code is always executed at the same frequency, the constraint is

- $T_1 = a \cdot T_2 = b \cdot T_3$ with $a, b \in \mathbb{N}, 1 < a < b$
  that is all code pieces must run at integer multiple frequencies of the fastest.

- $T_{ex1} + T_{ex2} + T_{ex3} < min\{T_1, T_2, T_3\}$
  Considering a worst case execution time of all code pieces.

In this chapter it will be shown mainly how the *Slaves* chain is handled by the *Master* using low level C programming configuring every needed peripheral and with some algorithms, while something will be omitted for sake of simplicity.

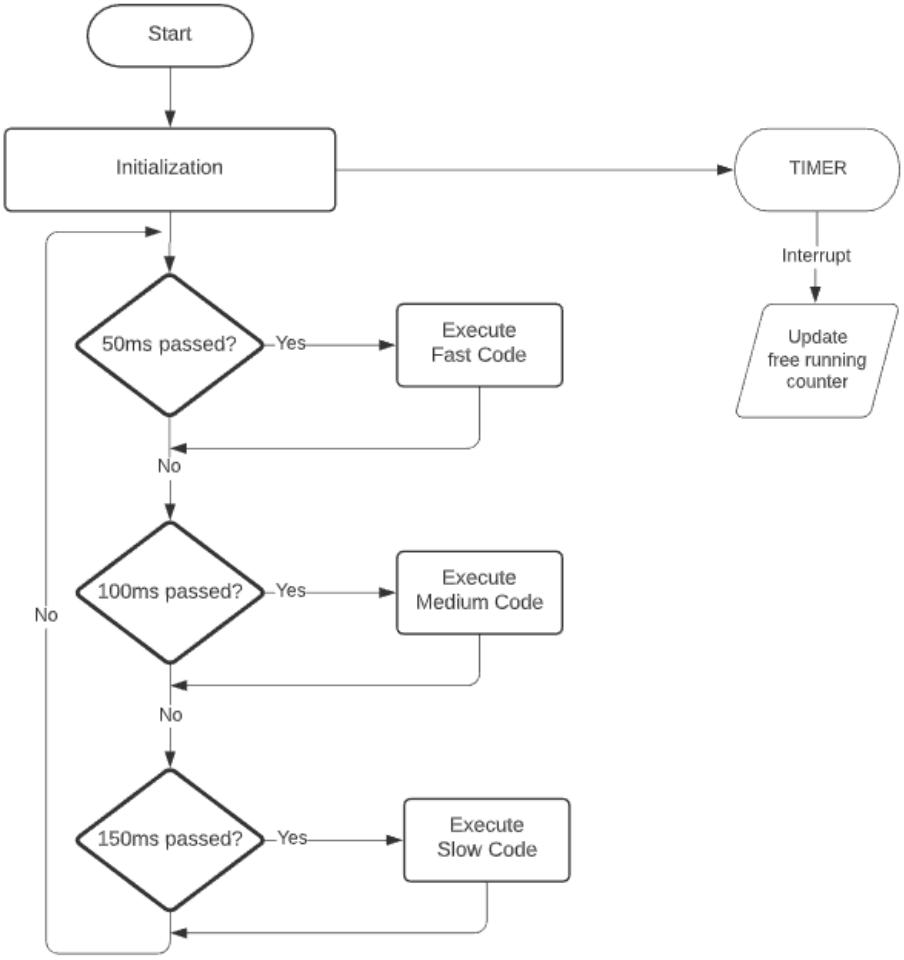

Figure 5.1: Firmware flow chart

5.2 Chain communication

5.2.1 UART driver and functions

The first thing to do in order to be able to communicate is configuring the UART peripheral and creating functions that can send and receive some messages (driver file in appendix C). The initialization function (line 3) first disable the clock gating, then selects the wanted clock source for the peripheral and writes to the (possibly multiple) configuration registers. This procedure is almost the same for every peripheral initialization (or however when configuration registers are changed). It is really important when working with peripheral that are connected with other components that their interface is interrupted when changing the configuration, in order to avoid unwanted damages. The peculiarity of this initialization is that it could be necessary to call it more than once, due to the fact that the *Slave* unit at start up works with a 250 kbit/s baud rate, and in order to configure a faster baud rate it must be communicated at this speed. After this, the function can be called again to change the baud rate to the desired one. The baud rate is selected by passing an argument to the function. This argument is used to divide the clock source of the UART peripheral in order to match the bit speed (line 10).
Sending a number of bytes via UART (line 18) is done by writing to the peripheral FIFO (line 27) in a blocking way (wait for it to be free if full at line 23). For receiving a number of bytes, the function in line 39 waits for every byte to arrive and saves it into a receiving buffer. A timer is started before the waiting *while* (line 45) that is set to trigger an interrupt after 1ms. If an error occurs and nothing is received, the interrupt sets a flag, *timeout*, and returns it to the caller.

5.2.2 Initialization and configuration

At start up, during the initialization phase in schematic 5.1, the chain is configured following the example of [20]. Communication is made of transaction frames composed of the following fields:

- **Frame initialization**, 1 Byte: If a command is issued from the *Master*, it

	7	6	5	4	3	2	1	0
Command Frame Init	FRM_TYPE = 1	REQ_TYPE			ADDR_SIZE	DATA_SIZE		
Response Frame Init	FRM_TYPE = 0	RESP_BYTES-1						

contains which type of request is made, how many data bytes will be sent, register addressing mode (1 or 2 byte). The type of request can be Single Device, Group Device or Broadcast, being them with or without a following response.

If the message is a response from the chain, it contains how many bytes are to be expected;

- **Device Address or Group ID**, 1 Byte: specify who needs to receive the command or who is responding;
- **Register Address**, 1 or 2 Byte: specify the register addressed for writing the following data;
- **Data**: data to be written or returned;
- **CRC**, 2 Bytes: Cyclic Redundancy Code to be sent or received.

In appendix D the function (line 36) sends a command to the chain, parametrized by type of request *cmdtype* with a switch case statement. Two type definitions (line 7,13) give names to the command types and device registers addresses. The buffer *msgbufsend* is filled with data to send depending on the command, while the data to be written inside device register is written, before calling the function, in *TXbuf*. CRC is calculated on every byte in the buffer with the function *CRC16_calc* and then appended to it. The buffer is finally passed to the UART function for sending it.

This initial configuration is a collection of call of this function, assigning to the devices parameters such as address, sampling mode and time for sensed cells, how many connected cells, etc. In order to keep the firmware flexible, the structure in line 3 is used to create an array of bytes (line 30) indicating how many devices the intended configuration is for, and for each device how many cell voltages and temperature sensors are connected to it.

5.2.3 CRC

The Cyclic Redundancy Check to implement with the BQ76PL455 uses a IBM polynomial (0x8005). The microcontroller offers a CRC unit able to calculate a 16 or 32 bit CRC in just two clock cycles. In appendix E the driver does a one-time

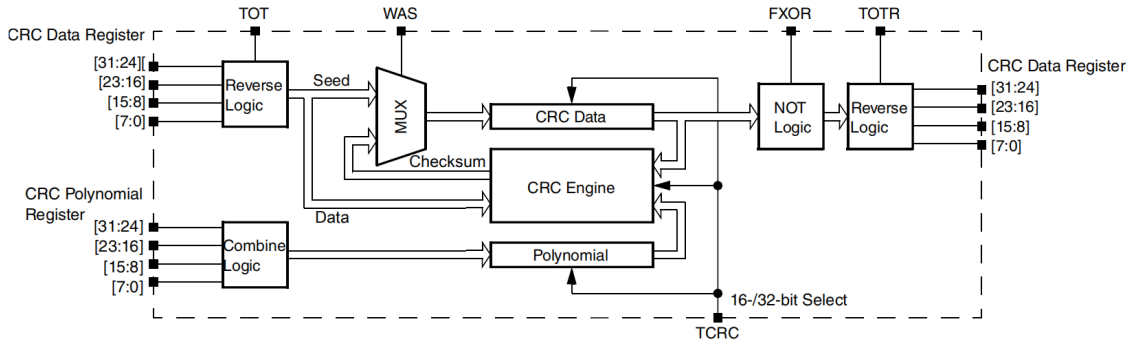

Figure 5.2: CRC unit schematic

initialization *CRC16_init* configuring the unit for a 16 bit protocol transposing all bits entering and exiting (required by IBM CRC) and writing the polynomial to the dedicated register. There are two ways offered for CRC calculation. "Burst" mode by calling the function *CRC16_calc* writes the seed with the function *CRC_seed_init* and then writes the data whose CRC need to be calculated in the data register for the specified number of times, both passed as argument to the function; at the end the function returns the 16 bit value by swapping upper and lower byte, required by BQ76PL455. In the "continuous" mode the application must call *CRC_seed_init* at the beginning, then *CRC_intermediate* how many times needed to write data byte wise, and must retrieve the result with *CRC_result*. The former method is useful for example when computing CRC for entire buffers, while the latter can be used for computing intermediate result between the reception of different bytes, as it is shown in the next section.

5.2.4 Sampling the values

A sampling request must be done in order to sample the configured number of channels (cells and temperature). This can be issued addressing single devices or

group of devices (if configured) or broadcasting. The last method is the quickest because it takes only one transaction from the *Master* to the *Slaves* via UART. By broadcasting the sampling request, all devices will sample and send the values

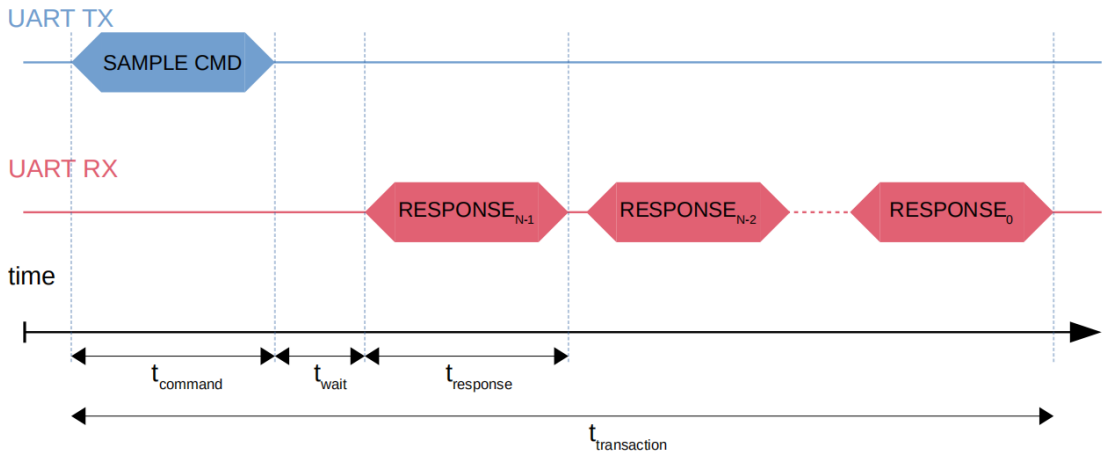

Figure 5.3: Sampling request transaction

starting from the last one. A rough estimation on how much time this will take id given by the formula

$$T_{transaction} = T_{command} + T_{wait} + N \cdot T_{response}$$

Where N is the number of devices. Each contribution will be analysed.

- $T_{command} = 5bytes \cdot t_{byte}(UART)$, where $t_{byte}(UART)$ is time needed for a byte to be transmitted on UART, i.e. 10 $bit \cdot t_{bit}$ depending on selected transmission speed;
- $T_{wait} = 2 \cdot t_{byte}(BQ) * (N-1) + T_{sample}$ that accounts for the transmission of the command up to the last device, sampling time of all signals and back to the first device. $t_{byte}(BQ)$ is fixed to 2.5μs/byte.
- $T_{response} = t_{byte}(UART) \cdot [2 \cdot (X_i + Y_i) + 3]$, where X_i, Y_i are each device sampled cell and temperature channel (2 bytes each) and the term 3 accounts for 1 byte header and 2 bytes CRC.

As an example, for N=12, X=12, Y=6, UART speed = 500 kbps, $T_{sample} = 100\mu s$ $T_{transaction} \approx 11.3ms$. This is an estimation, due to the fact that it is not possible

to know a priori the exact communication timing of chained devices (apart from transmission speed). It is important to measure this time in the actual implementation in order to establish the frequency at which the *Master* can measure the entire system.

Sampling cells voltages and temperatures is the most critical task of the system, since the actions to be taken depends on them. After receiving all values, calculations on data need to be done, further increasing cycle time. What can be done is try to manipulate the received data before the next one arrives, considering that, in the worst case, at an UART baud rate of 1Mbps one byte takes 10μs to be received. The proposed algorithm in appendix F uses previously described drivers to cycle the reception of each device response frame. The function (line 18) returns a boolean value, true if an error occur either because the chain did not respond (line 38) or due to non correspondent CRC (line 97). For every device (line 46) first all cell voltages are received (line 48), then all temperatures (line 70). Some structures are used in order to store values:

```
1     struct VCELL{
2             uint16t cellH;
3             uint16t cellL;
4             uint16t cellAvg;
5             uint08t cellHid;
6             uint08t cellLid;
7     };
8
9     struct TCELL {
10            sint16t tempH;
11            sint16t tempL;
12            sint16t tempAvg;
13            uint08t tempHid;
14            uint08t tempLid;
15    };
16
17    struct TDEBUG{
18            uint16t tmux;
19            sint16t t1;
```

```
20              sint16t t2;
21              sint16t t3;
22      };
23
24      struct VDEBUG{
25              uint16t vmux;
26              uint16t v1;
27              uint16t v2;
28              uint16t v3;
29      };
30
31      struct SYS_INFO{
32          uint16t TotalPackV;
33          sint16t Current;
34          uint16t status;
35          uint16t SOC;
36      };
```

VCELL contains the highest and lowest cell voltage, together with their ID (i.e. position in the chain), together with average value. The same is done for temperatures in *TCELL*. These structures are pointed by the CAN driver (not treated) so that they are sent as messages of 8 bytes each. The information provided by these structures can be enough for an external ECU in order to take decisions, as they report the worst cell conditions. Sending all temperature and voltage readings can be too taxing on the CAN bus, so structures *TDEBUG* and *VDEBUG* are used to send only 3 values at each iteration together with a multiplexing value in order to identify their ID. In *SYS_INFO* the total pack voltage is obtained as sum of every single cell, and is also stored current sensor value and SoC. The *status* variable will be described in another section.

5.3 Temperature conversion algorithm

The *Slave* unit will send the sampled value of the NTC in the configuration seen in 3.7. From measured voltage, the resistance value need to be calculated, and then converted to temperature. For simplicity we will call R_p the pull up resistor, R_0 the NTC resistance value at $T_0 = 25°C$, R the resistance of the NTC at temperature T to be measured, V_{DIG} the value in LSB of the ADC measure received from the *Slave*, V_{FSR} the ADC full scale range in Volts, N_b the ADC number of bits

$$V_{DIG} = V_{FSR}\frac{R}{R+R_p}\frac{2^{N_b}-1}{V_{FSR}} = \frac{R}{R+R_p}(2^{N_b}-1) \tag{5.1}$$

Then R can be calculated as:

$$R = R_p\frac{V_{DIG}}{2^{N_b}-1-V_{DIG}} \tag{5.2}$$

From the NTC characteristic equation we can get temperature from resistance by inverse formula of 3.1:

$$T = \frac{1}{\frac{1}{T_0}+\frac{1}{\beta}ln(\frac{R}{R_0})} \tag{5.3}$$

By substituting 5.2 into 5.3 we obtain:

$$T = \frac{1}{\frac{1}{T_0}-\frac{1}{\beta}ln(\frac{R_0}{R_p}(\frac{2^{N_b}-1}{V_{DIG}}-1))} \tag{5.4}$$

In case $R_p = R_0$ the term $\frac{R_0}{R_p}$ can be eliminated. The temperature obtained is Kelvin, so a quantity of 273 should be subtracted for transforming it in °C.

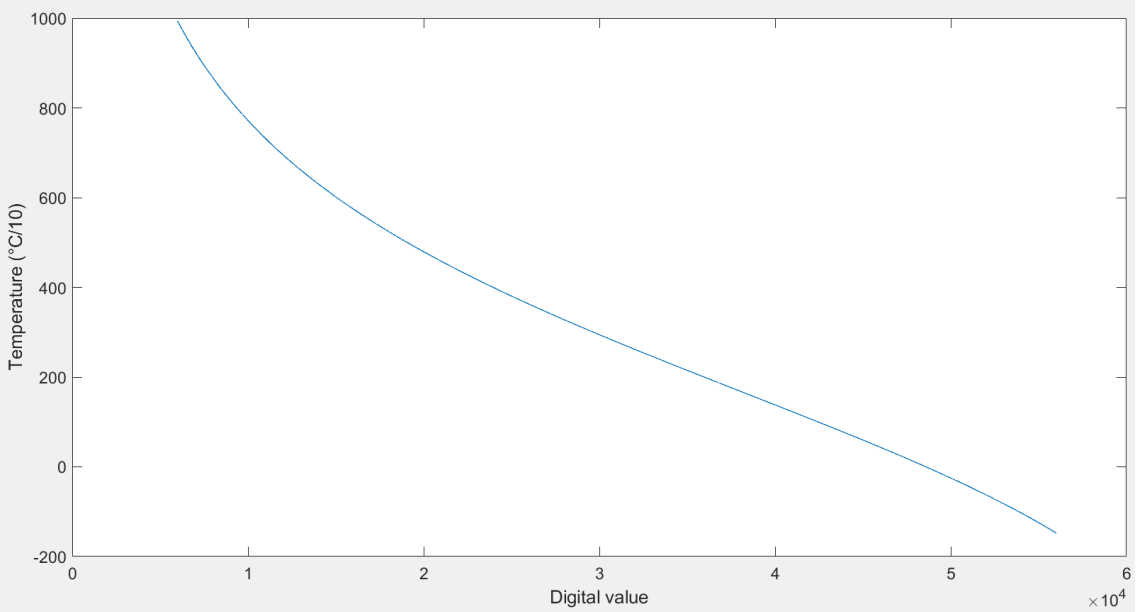

Figure 5.4: Plot of equation 5.4

The equation could be implemented as it is by using a recursive method for approximating the natural logarithm, but it would be too slow for using it between the reception of two consecutive voltage measurements as done in 5.2. A first approximation can be done by fixing a number of points, let's say 20, and obtain a linear piecewise function. To further improve speed, calculations can be done as the integer unit of the microcontroller would do, by truncating the decimal part, obtaining the function in 5.5. As not to lose too much precision, equation 5.4 has 10 at nominator, so only the second decimal digit is lost.

A piecewise approximation is more accurate when the slope of the function is closer to 0. The biggest error is indeed from 70 to 100°C, which is maximum 0.8°C. Error can be decreased if more points are considered when linearizing the function. Considering the intrinsic inaccuracy of an NTC measuring process, this error is acceptable, also because is a worst case estimation for normal use, as the temperature sensed is higher. These calculations have been carried out with a Matlab script (A). How does this approximation translates in the code is shown in B. The temperature value of 21 equally spaced points is saved in the vector *y[i]* (two bytes each), from a

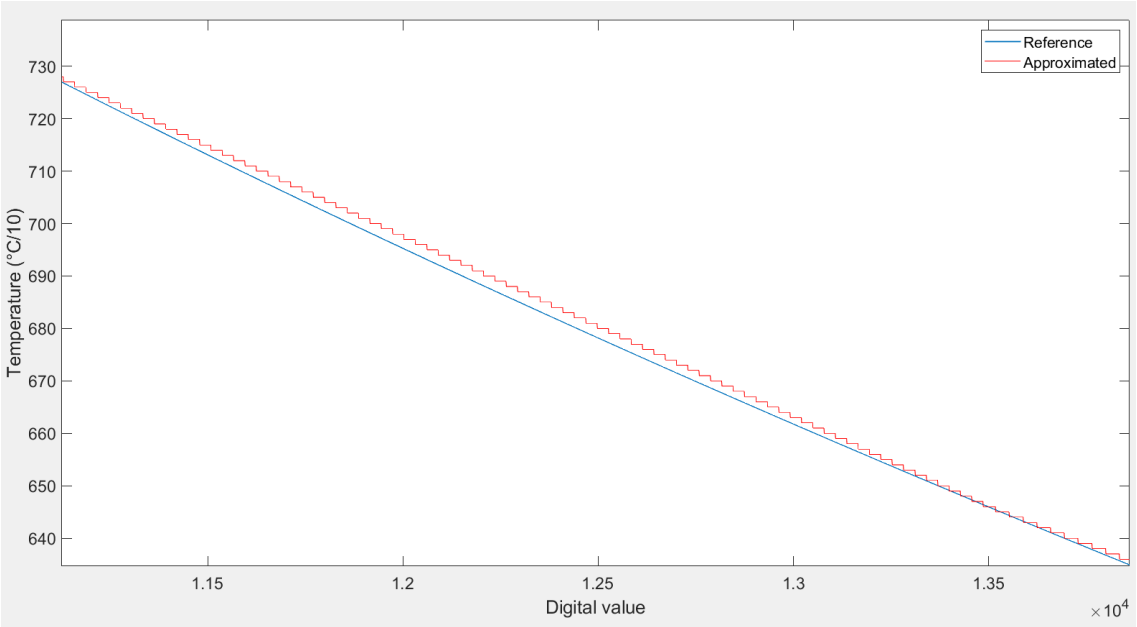

Figure 5.5: Piecewise integer approximation

digital value of 6000 to 56000, i.e. 99°C to -15°C. The function called converts the unsigned value passed to it into a signed temperature value through the formula at line 13.

5.4 Error Detection Mechanism

One of the BMS role is to inform the motor controller of its operational status (contactors management, errors detection, etc.) and cell status. In order to this, a *status* variable of two bytes lenght is used. Each bit corresponds to the presence/absence of a specific error. After collecting data (see previous sections) each value is compared to predetermined thresholds, but they must exceed them several times before of resulting in an error. For example, if values are sampled every 10ms and the system need to react (i.e. open/close contactors) every 100ms, it can be useful to wait for the 10^{th} error to be detected before opening the contactors. Error bits can be sent to the CAN bus in order to tell the motor controller to interrupt current flow as soon as possible. If the error persists for more than 100ms, contactors will be opened anyway in order to avoid battery pack damages. These information can also be used to let the vehicle users know battery problems.

Chapter 6

Conclusion

In order to evaluate the proposed work, a test bench has been created, composed of a small Li-Ion battery pack in 12S1P made from Sony VTC6 cells monitored from two evaluation boards of the *BQ76PL455A* [21] connected in chain with an evaluation board of the microcontroller described in the previous chapter [22]. These evaluation modules offer the possibility of creating a system without the need of hardware development, reducing cost and saving time, at the compromise of having an hardware platform that is the most generic possible, i.e. not tailored for any specific application. The goal was to establish a stable communication with the chain of monitoring devices, exploiting all of their functions, thanks to the use of all peripherals as previously described. The microcontroller board offers a CAN bus transceiver, to which a CAN to USB device has been connected from a PC.
The CAN database shown in figure 6.1 is used to send commands and receive information through a dedicated software on the PC that simulates a CAN node.

	Name	ID HEX	Frame Format	DLC	TX Node	Comment
1	BMS_VCELL	080	Standard	8	BMS	Cell V (low, high, avg)
2	BMS_TCELL	082	Standard	8	BMS	Cell T (low, high, avg)
3	BMS_SYS_INFO1	084	Standard	8	BMS	BMS system information
4	BMS_CMD	07e	Standard	3	VMU	Received command from external VMU
5	BMS_DEBUG_ALL_T	102	Standard	8	BMS	Single thermistor temperature (multiplexed).
6	BMS_DEBUG_ALL_V	104	Standard	8	BMS	Single cell voltage (multiplexed).

Figure 6.1: CAN database file

Remarking the scope of the thesis, the objective was to lay the foundation for an energy management system by an extensive analysis of various hardware possibilities and choosing one to develop. The result is a multi slave single master architecture. The first stage of development consisted in using evaluation modules hardware for both *Master* and *Slave* units building a simple test scenario with a small battery pack. The work then consisted in conceiving a firmware architecture that:

- has an extensive library for *Slave* settings and communication
- Fully exploits hardware microcontroller peripherals for computation and communication (CRC, UART, CAN) for maximum flexibility and efficiency
- Cyclically check and intervene on battery safety components to keep safe operating status and inform other ECUs on the CAN bus.

Appendix A

Matlab script of temperature approximation

```
N = 19;
stp_x = 50000/(N+1);
BETA = 3428;
x1 = 6000:1:56000;
y1 = 10*(1./(1/298 - (1/BETA)*log((65535./x1)-1)) - 273);
    %%NTC equation gives 0.1 C as unit of measure

x2 = 6000:stp_x:56000;
v=round(interp1(x1,y1,x2, 'linear')); %%rounded linear
   interpolation of the function

for i=1:N+1
    stp_y(i)= v(i+1)-v(i);
end

y2 = 0:1:50000;

for i=1:50000
    y= x1(i) - 6000;
    rem = floor(y/stp_x);
    y2(i) = v(rem+1) + round(round(stp_y(rem+1)*(y-rem*
       stp_x))/stp_x);
end
```

```
y2(50001) = v(21);
e = y2-y1;
m = max(e)
figure(1)
plot(x1,y1, 'LineWidth', 1)
hold on
plot(x1,y2, 'r')
xlabel('Digital value')
ylabel('Temperature ( C/10)')
set(gca,'FontSize',14);
legend('Reference','Approximated')

figure(2)
plot(x1,e, '.')%%, 'LineStyle', 'dot')
xlabel('Digital value')
ylabel('Absolute error ( C/10)')
```

Appendix B

Temperature algorithm C code

```
#define TOFFSET = 6000
#define TSTEP = 2500
uint16t y[21] = {993, 841, 732, 646, 575, 513, 458, 409, 363, 319, 278, 238,
↪  200,161, 123, 84, 44, 2, -43, -92, -147};

sint16t Get_T(uint16t value)
{
        uint08t i;
        sint32t temp;

        if((value > 5999) && (value < 56000))
        {
                i = (value - TOFFSET)/TSTEP;
                temp = y[i]+(value - TOFFSET - i*TSTEP)*(y[i+1]-y[i])/TSTEP;
                return (sint16t) temp;
        }
        else if(value < 6000)
        {
                return -250; /*min value saturation*/
        }
        else
        {
                return 800; /*max value saturation*/
        }

}
```

Appendix C

UART driver

```
1  boolt timeout;
2
3  void UARTinit(uint08t speed_sel) /*1=1M, 2=500k, 4=250k*/
4  {
5
6          PCC->PCCn[PCC_LPUART0_INDEX] &= ~PCC_PCCn_CGC_MASK;    /* Ensure clk
           ↪  disabled for config */
7            PCC->PCCn[PCC_LPUART0_INDEX] |= PCC_PCCn_PCS(6u)    /* Clock Src= 6
             ↪  (SPLLDIV2_CLK) */
8                                         |  PCC_PCCn_CGC_MASK;     /* Enable clock
                                          ↪  for LPUART1 regs */
9
10           LPUART0->BAUD = 0x0FA00000 & (0x05 << speed_sel);  /* Initialize for
             ↪  selected baud, 1 stop bit, 8 data bits: */
11
12           LPUART0->CTRL=0x000C0000;    /* Enable transmitter & receiver, no
             ↪  parity, 8 bit char: */
13
14 }
15
16 //function for sending a vector of bytes
17
18 void UARTsend(uint08t *data, uint08t size)
19 {
20
21         for(uint08t i=0; i<size; i++)
22         {
23                 while((LPUART0->STAT &
                   ↪  LPUART_STAT_TDRE_MASK)>>LPUART_STAT_TDRE_SHIFT==0)
24                         {
25                                 //wait for fifo to be empty;
26                         }
27                 LPUART0->DATA= *(data+i);
```

```
28          }
29
30  }
31
32  /*UART receive function
33   * Parameters:
34   * 1) "data" for storing the received characters
35   * 2) "size" for how may characters to be received
36   * 3) "offset" for storing characters in "data"
37   */
38
39  boolt UARTreceive(uint08t data[], uint08t size, uint08t offset){
40          timeout = BOOL_FALSE;
41
42          for(uint08t i=0; i<size ; i++){
43                  LPIT0->TMR[1].TCTRL |=  LPIT_TMR_TCTRL_T_EN(1); /*enable the
                    ↪  timer */
44
45                  while(((LPUART0->STAT &
                    ↪  LPUART_STAT_RDRF_MASK)>>LPUART_STAT_RDRF_SHIFT==0) && timeout
                    ↪  == BOOL_FALSE)
46                  {
47                          //wait for a byte to be received, timeout expires in 1ms
48                  }
49
50                  LPIT0->TMR[1].TCTRL &= ~LPIT_TMR_TCTRL_T_EN(1); /*disable the
                    ↪  timer */
51
52                  if(timeout == BOOL_FALSE){
53                          data[offset+i] = LPUART0->DATA;
54                  }
55                  else{
56                      return timeout;
57                  }
58
59          }
60          return timeout;
61  }
```

Appendix D

Sending a Command to the chain

```
#define DevNum 2

typedef struct{
        uint08t DevCell; uint08t DevTemp;
}BMS_slaveconfig_t;

typedef enum{
        SingleNoResp = 0x9U, SingleResp = 0x8U,
        BroadcastNoResp =0xFU, BroadcastResp = 0xEU,
        GroupNoResp = 0xBU,         GroupResp = 0xAU,
} BMS_msg_cmd_t;

typedef enum{
        CMDreg = 0x2U, CHANNELSreg = 0x3U,
        OVERSAMPLEreg = 0x7U, DEVADDRreg = 0xAU,
        DEVCTRLreg = 0xCU, NCHANreg = 0xDU,
        DEVCONFIGreg = 0xEU, COMCONFIGreg = 0x10U,
        BALCONFIGreg = 0x13U, BALENreg = 0x14U,
        SAMPLEDELAYreg = 0x3DU, CELLSPERreg = 0x3EU,
        AUXSPERreg = 0x3FU, STATUSreg = 0x51U,
        FAULTSUMMARYreg = 0x52U, FAULTUVreg = 0x54U,
        FAULTOVreg = 0x56U, FAULTAUXreg = 0x58U,
        CELLUVreg = 0x8EU, CELLOVreg = 0x90U,
        AUXOUVreg = 0x92U, //offset for AUX1, AUX2 etc. is 4 up to AUX7UV
        AUXOOVreg = 0x94U, //offset for AUX1, AUX2 etc. is 4 up to AUX7OV
        CELLOFFSETreg = 0xD2U, CELLGAINreg = 0xD3U,
        AUXOOFFSETreg = 0xD4U, //offset for AUX1, AUX2 etc. is 2
} BMS_registers_t;

BMS_slaveconfig_t BMSslaves[DevNum] = { //for every slave set how many cells/temp
↪  it monitors, 1st is bottom of stack
        {6,8}, /*1st*/
        {6,8}  /*2nd*/
```

```
};

uint08t TXbuf[8];
void BMS_send_cmd(BMS_msg_cmd_t cmdtype, uint08t devaddr, BMS_registers_t
↪  regdest, uint08t datasize, uint08t respsize)
{
        static uint08t i;
        static uint16t tempcrc;
        static uint08t msgbufsend[8];

        msgbufsend[0] = ((uint08t) cmdtype << 4) + datasize;

        switch (cmdtype)
        {
                case SingleNoResp: //device address is sent
                {
                        msgbufsend[1] = devaddr;
                        msgbufsend[2] = (uint08t) regdest;
                        for (i=0; i<datasize; i++)
                        {
                                msgbufsend[3+i] = TXbuf[i];
                        }
                        tempcrc = CRC16_calc(&msgbufsend[0], 3+datasize);
                        msgbufsend[3+datasize] = tempcrc >> 8;
                        msgbufsend[3+datasize+1] = tempcrc;
                        UARTsend(&msgbufsend[0], 5+datasize);
                        break;
                }
                case SingleResp: //device address and response size is sent
                {
                        msgbufsend[1] = devaddr;
                        msgbufsend[2] = (uint08t) regdest;
                        msgbufsend[3] = respsize-1;
                        tempcrc = CRC16_calc(&msgbufsend[0], 4);
                        msgbufsend[4] = tempcrc >> 8;
                        msgbufsend[5] = tempcrc;
                        UARTsend(&msgbufsend[0], 6);
                        break;
                }
                default: //if broadcast type, no device address is sent
                {
                        msgbufsend[1] = (uint08t) regdest;

                        for (i=0; i<datasize; i++)
                        {
                                msgbufsend[2+i] = TXbuf[i];
```

```
78                          }
79                          tempcrc = CRC16_calc(&msgbufsend[0], 2+datasize);
80                          msgbufsend[2+datasize] = tempcrc >> 8;
81                          msgbufsend[2+datasize+1] = tempcrc;
82                          UARTsend(&msgbufsend[0], 4+datasize);
83                          break;
84                  }
85          }
86
87  }
```

Appendix E

CRC driver

```
#define CRC16_IBM_POLY (0x8005)

void CRC16_init(void){
        PCC->PCCn[PCC_CRC_INDEX] |= PCC_PCCn_CGC_MASK;        /* enable CRC clock
        ↪ */

        CRC->CTRL |= CRC_CTRL_TCRC(0);        /* enable 16-bit CRC protocol */
        CRC->CTRL |= CRC_CTRL_TOT(2);        /* all bits transposed in write */
        CRC->CTRL |= CRC_CTRL_TOTR(2);        /* all bits transposed in read*/
        CRC->CTRL |= CRC_CTRL_FXOR(0);        /* no XOR on reading. */
        CRC->GPOLY = CRC16_IBM_POLY;
}

void CRC_seed_init(void){
        CRC->CTRL |= CRC_CTRL_WAS_MASK;        /* Set to program the seed value.
        ↪ */
        CRC->DATAu.DATA = 0x0000;        /* seed value */
        CRC->CTRL &= ~CRC_CTRL_WAS_MASK;        /* Clear to start writing data
        ↪ values. */
}

void CRC_intermediate(uint08t *data){
        CRC->DATAu.DATA_8.LL = *data;
}

uint16t CRC_result(void){
    uint16t temp_crc;
    temp_crc = ((uint16t)CRC->DATAu.DATA_8.HL << 8) + CRC->DATAu.DATA_8.HU;
    return temp_crc;
}

uint16t CRC16_calc(uint08t *data, uint08t size){
        uint16t temp_crc;
```

```
31          CRC_seed_init();
32          for(uint08t i=0; i < size; i++)        {
33                  CRC->DATAu.DATA_8.LL = *(data+i);        /* write data values */
34          }
35          temp_crc = ((uint16t)CRC->DATAu.DATA_8.HL << 8) + CRC->DATAu.DATA_8.HU;
            ↪  /*bytes are swapped for BQ76PL455*/
36          return temp_crc;
37  }
```

Appendix F

Chain sampling function

```
1  /*structure are used to pass the values for sending via CAN*/
2  struct SYS_INFO BMS_SYS_INFO;
3  struct TCELL BMS_TCELL;
4  struct VCELL BMS_VCELL;
5
6  uint08t RXbuf[3]; /*UART RX buffer*/
7  uint08t TXbuf[8]; /*UART TX buffer*/
8  uint16t allV[12]; /*store all V, used for debug*/
9  sint16t allT[18]; /*store all T, used for debug*/
10
11 /*
12  * Function that issue the sampling for all devices and receive the sampled
   ↪  values
13  * Between reception of values on UART, all operations including CRC are
   ↪  performed in order to eliminate overhead
14  * Since CRC is calculated with intermediate values, driver function 'CRC16calc'
   ↪  is not used because we don't want
15  * to reset the seed nor to read the intermediate value
16  * TRUE = error, FALSE = ok
17  */
18 boolt BMS_get_values(void){
19         //local var
20         static uint16t tempoVar; //used to combine 2 bytes
21         static uint32t cellV;
22         static sint16t cellT;
23         static boolt locRet;
24
25         /*
26          * START
27          */
28
29         //send sampling request for every device starting from top, manage data
           ↪  before the reception of the next one
```

```
30          TXbuf[0] = DevNum-1;
31          BMS_send_cmd(BroadcastResp, 0, CMDreg, 1, 0);
32          CRC_seed_init();
33
34          //reset indexes
35          uint08t indexcell= SeriesCell, indextemp=TotalTemp;
36
37          //receive 1st header
38          if(UARTreceive(RXbuf, 1, 0) == 0){ //no timeout
39                  //reset variables
40                  uint32t temptotalcellV = 0;
41                  sint32t totalcellT = 0;
42                  BMS_VCELL.cellH = 0, BMS_VCELL.cellL = 6000;
43                  BMS_TCELL.tempH = -300, BMS_TCELL.tempL = 6000;
44                  CRC_intermediate(&RXbuf[0]);
45
46                  for(uint08t index=DevNum; index>0; index--){
47                          //cell voltages
48                          for(uint08t i=BMSslaves[index-1].DevCell; i>0;
                            ↪  i--)        {
49                                  (void)UARTreceive(RXbuf, 2, 0);
50                                  tempoVar = ((uint16t)RXbuf[0] << 8) |
                                    ↪  (uint16t)RXbuf[1]; //16-bit ADC value of cell
                                    ↪  voltage
51                                  cellV = (tempoVar * 5000) / 65535; //cell voltage
                                    ↪  in mV
52                                  CRC_intermediate(&RXbuf[0]);
53                                  CRC_intermediate(&RXbuf[1]);
54                                  indexcell--;
55                                  //check if it is the currently highest/lowest
                                    ↪  cell voltage
56                                  if(cellV > BMS_VCELL.cellH){
57                                          BMS_VCELL.cellH = (uint16t) cellV;
58                                          BMS_VCELL.cellHid = indexcell;
59                                  }
60                                  else if(cellV < BMS_VCELL.cellL){
61                                          BMS_VCELL.cellL = cellV;
62                                          BMS_VCELL.cellLid = indexcell;
63                                  }
64                                  else{}
65                                  temptotalcellV += cellV;
66                                  allV[indexcell] = cellV;
67                          }
68
69                          //cell temperatures
70                          for(uint08t i=BMSslaves[index-1].DevTemp; i>0; i--){
```

```
71                              (void)UARTreceive(RXbuf, 2, 0);
72                              tempoVar = ((uint16t)RXbuf[0] << 8) | RXbuf[1];
                                ↪  //16-bit ADC value of NTC voltage
73                              cellT = Get_T(tempoVar); //cellT contains the
                                ↪  signed temperature in 0.1C
74                              CRC_intermediate(&RXbuf[0]);
75                              CRC_intermediate(&RXbuf[1]);
76
77                              indextemp--;
78                              //check if it is the currently highest/lowest
                                ↪  cell temperature
79                              if(cellT > BMS_TCELL.tempH){
80                                      BMS_TCELL.tempH = cellT;
81                                      BMS_TCELL.tempHid = indextemp;
82                              }
83                              else if(cellT < BMS_TCELL.tempL){
84                                      BMS_TCELL.tempL = cellT;
85                                      BMS_TCELL.tempLid = indextemp;
86                              }
87                              else{}
88                              totalcellT += cellT;
89                              allT[indextemp] = cellT;
90                      }
91
92                      //receive CRC
93                      (void)UARTreceive(RXbuf, 2, 0);
94
95                      if((RXbuf[0] == CRC->DATAu.DATA_8.HL) && (RXbuf[1] ==
                        ↪  CRC->DATAu.DATA_8.HU)){ //check CRC
96                      }
97                      else{
98                              locRet = BOOL_TRUE; //wrong CRC
99                      }
100
101                     if(index != 1){
102                             //receive next header
103                             UARTreceive(RXbuf, 1, 0);
104                             CRC_seed_init();
105                             CRC_intermediate(&RXbuf[0]);
106                     }
107                     else{}
108             }
109
110             if(locRet == BOOL_FALSE){
111                     BMS_VCELL.cellAvg = (uint16t)
                        ↪  (temptotalcellV/SeriesCell);
```

```
112                         BMS_TCELL.tempAvg = (sint16t) (totalcellT/TotalTemp);
113                         BMS_SYS_INFO.TotalPackV = (uint16t) (temptotalcellV/100);
                            ↪  //0.1V format
114                 }
115                 else{}
116         }
117         else{
118                 locRet = BOOL_TRUE; //timeout
119         }
120         return locRet;
121 }
```

Acronyms

μC microcontroller. 14, 17

AFE Analog Front-End. 18

BCU Battery Control Unit. 11

CAN Controller Area Network. 43

CC Coulomb Counting. 47

DSP Digital Signal Processor. 50

EMC Electromagnetic Compatibility. 45

EV Electric Vehicles. 10, 11

FPGA Field Programmable Gate Array. 50

Grounded Low Voltage System GLVS. 41

HEV Hybrid Electric Vehicles. 10

HMI Human-Machine Interface. 30

HV High Voltage. 10, 11

HVD High Voltage Disconnect. 31

HVIL High Voltage Interlock. 31

IC integrated circuit. 17

ICE Internal Combustion Engine. 10

IMD Isolation Monitoring Device. 41

ISO International Organization for Standardization. 9, 10

ISR Interrupt Service Routine. 54

LIBs Lithium Batteries. 9

LV Low Voltage. 10, 11

NiMH Nickel-Metal Hydride. 9

NTC Negative Temperature Coefficient. 21

OCV Open Circuit Voltage. 48

OpAmp Operational Amplifier. 33, 35

PCB printed circuit board. 13

PHEV Plug-In Hybrid Electric Vehicles. 10, 11

PWM Pulse Width Modulated. 41

SAR Successive Approximation Register. 17

SoC State-of-Charge. 11, 38, 47

SoH State-of-Health. 11, 38, 47

TVS Transient Voltage Suppressor. 25

UART Universal Asynchronous Receiver Transmitter. 17

VMU Vehicle Management Unit. 30

Bibliography

[1] Krunal Maniar. Comparing shunt- and hall-based isolated current-sensing solutions in hev/ev. http://www.ti.com/lit/an/sbaa293b/sbaa293b.pdf, 2018.

[2] Introduction to the controller area network. http://www.ti.com/lit/an/sloa101b/sloa101b.pdf, 2002.

[3] Y. Jeong, Y. Cho, J. Ahn, S. Ryu, and B. Lee. Enhanced coulomb counting method with adaptive soc reset time for estimating ocv. In *2014 IEEE Energy Conversion Congress and Exposition (ECCE)*, pages 1313–1318, Sep. 2014.

[4] M. S. Halper and J. C. Ellenbogen. *Supercapacitors: A Brief Overview*. The MITRE Corporation, 2006.

[5] UNECE. rule n° 100, rev. 2, 12 August 2013.

[6] ISO. 6469-3, 2011-12-01.

[7] ISO. 12405-2, 2012.

[8] https://www.diodes.com/assets/Datasheets/ZXTN4004K.pdf.

[9] http://www.ti.com/lit/ds/symlink/iso7742.pdf.

[10] https://www.te.com/commerce/DocumentDelivery/DDEController?Action=srchrtrv&DocNm=5-1773450-5_sec7_LEV200&DocType=CS&DocLang=English.

[11] https://www.ti.com/lit/ds/symlink/amc1311.pdf.

[12] https://www.ti.com/lit/ds/symlink/sn6501.pdf.

[13] https://www.lem.com/sites/default/files/products_datasheets/dhab_s_157_public_datasheet.pdf.

[14] https://www.bender-it.com/fileadmin/content/Products/d/e/IR155-32xx-V004_D00115_D_XXEN.pdf.

[15] Ole-Kristian Skroppa and Scott Monroe. Common mode chokes in can networks: Source of unexpected transients, 2008.

[16] Jiuchun Jiang and Caiping Zhang. *Fundamentals and Applications of Lithium-Ion Batteries in Electric Drive Vehicles*. Wiley, 2015.

[17] P. A. Topan, M. N. Ramadan, G. Fathoni, A. I. Cahyadi, and O. Wahyunggoro. State of charge (soc) and state of health (soh) estimation on lithium polymer battery via kalman filter. In *2016 2nd International Conference on Science and Technology-Computer (ICST)*, pages 93–96, Oct 2016.

[18] P. Shen, M. Ouyang, L. Lu, J. Li, and X. Feng. The co-estimation of state of charge, state of health, and state of function for lithium-ion batteries in electric vehicles. *IEEE Transactions on Vehicular Technology*, 67(1):92–103, Jan 2018.

[19] `http://www.ti.com/lit/ds/symlink/hdc2021.pdf`.

[20] Stephen Holland. bq76pl455a-q1 software design reference, 2014.

[21] `http://www.ti.com/lit/ug/sluuba7a/sluuba7a.pdf`.

[22] `https://www.nxp.com/docs/en/quick-reference-guide/S32K144EVB-QSG.pdf`.