POLITECNICO DI TORINO

Master of Science Degree in Electronic Engineering

Masters Thesis

Electronic architecture of a Formula Student Electric Car

Supervisor:
Prof. Marcello Chiaberge

Candidate:
Federico Porrá

March 2020

Table of contents

Chapter 1

Introduction

1.1 Intent

The purpose of this thesis is the architectural analysis of an electrical Formula Student car that competed in the 2019 season manufactured by Squadra Corse from Polytechnic of Turin. For simplicity this document will treat only the electronic management units that for rules impositions have to be mounted in the vehicle and for the normal vehicle's operation. At the beginning a little overview of Formula Student's event will be given in order to explain the environment in which the project took place. Rules restrictions and constraints will be analyzed and synthesized to create a set of requirements that will drive the design of the architecture.
This document will focus on three main parts of this Electric Vehicle: a data acquisition system, an accumulator management system and the safety system. For each module both hardware and software will be designed and the architecture of the previous vehicle will be examined focusing on problems that arises. The last part will deal with a custom graphical interface for in vehicle debugging purposes.

1.2 Overview

This document is divided into four sections: an Introduction (1), the State of Art (2), the Hardware part (3) and the Software part (4). The *State of Art* chapter will focus on what formula student is, analyzing the rules that will necessary drives the project's specifications, considering the previous year vehicle and creating a set of specifications and constraints that will drive the entire electronic design. The *Hardware* chapter will explain the main hardware feature starting from the choice of a microcontroller suitable for the intent, the design of a dedicated circuit for analog signal acquisition, and the schematic of a battery management system for low voltages purposes and finishing with the safety electronics systems mandatory due to rules. Here below is reported in figure 1.1 the final electronic architecture.

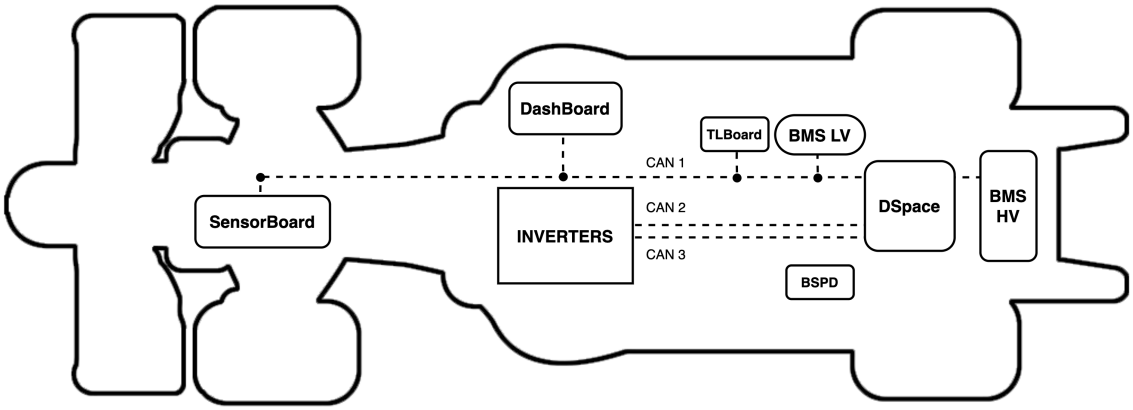

Figure 1.1: Final Electronic Architecture

The *Software* chapter will describe in detail the software architecture of the developed firmwares. The use of an open source bootloader will be justified describing a modular architecture that reduced the overall development time creating a robust system. Hence firmwares of all units will be described focusing on main features and a host software that runs on PC will be presented. This for debug and directly update firmwares on vehicle.

1.3 Author's contribution on SC19

Squadra Corse is internally divided into Divisions. Being a part of the Electric and Electronic division was an educational but also challenging experience. The division was composed by two members that co-worked for all the time; due to the reduced number of human resources the work has been done in cohesion and is difficult to underline who-did-what. My contribution relates to both *hardware* design, starting from the schematic to the layout of the board and *software* design. But also manufacturing the vehicle was a crucial part: each board has been electrically and functionally tested to ensure the correct behaviour. The wiring harness has been

developed and physically placed in-vehicle. The overall contribution to the project starts from the design stage and jumps into the manufacturing processes (PCBs, soldering components, testing, harness crimping).

Figure 1.2: Tems's photo in Varano de Melagari circuit

Chapter 2

State of Art

This chapter will lay the groundwork for the project specification starting from a deep analysis of the requirements in terms of performances and restrictions ending with a detailed architecture. Formula Student events will be presented: rules, scores, limitations and team's previous vehicle will be introduced and synthesized into a set of requirements that will drive the design of the vehicle participant 2019's season.

2.1 Formula Student

Formula SAE is a student design competition organized by SAE International. The concept behind Formula SAE is that a fictional manufacturing company has contracted a student design team to develop a small Formula-style race car. The prototype race car is to be evaluated for its potential as a production item. The target marketing group for the race car is the non-professional weekend autocross racer. Each student team designs, builds and tests a prototype based on a series of rules, whose purpose is both ensuring on-track safety (the cars are driven by the students themselves) and promoting clever problem solving. The prototype race car is judged in a number of different events divided in two classes: Static and Dynamic. Static events are compose by three different disciplines: Engineering Design, Business Plan and Cost and Manufacturing. Dynamic events are composed by four different disciplines: Acceleration, Skid Pad,Autocross and Endurance. For each discipline the following point are assigned:

Event	Points
Design	150
Business Plan	75
Cost and Manufacturing	100
Acceleration	75
Skipad	75
Autocross	100
Endurance	325
Efficiency	100

One of the most important discipline, from a technical point of view is the Engineering Design for which at least 15% of the total score can be gained. Different judges

will examinate more in details the project each one of them focusing into different area of interest. For what concern the vehicle electronics, team members belonging to Electronic area will be interviewed and judges will perform a series of questions in order to understand more in deep the motivations behind engineering choices. Even project and development methodologies will be analyses and evaluated. This static event is one of the most difficult since innovative techniques and solutions must be provided to reach the highest scoring. In addition to these events, various sponsors of the competition provide awards for superior design accomplishments. At the beginning of the competition, the vehicle is checked for rule compliance during the Technical Inspection. Its braking ability, rollover stability are checked before the vehicle is allowed to compete in the dynamic events. Formula SAE encompasses all aspects of a business including research, design, manufacturing, testing, developing, marketing, management, and fund raising. The volunteers for the design judging include some of the racing industry's most prominent engineers and consultants. Squadra Corse is the Formula Student team of Polytechnic of Turin, founded in 2003 and competing since 2005 with their first car, SC05. The first electric car was built in 2012, the SC12e. Today, the latest prototype is the 6th electric car built by the team, named SC19 (internally *Lucia*). The project of the vehicle is based on SCXV configuration with 4-wheel drive outboard. I was a part of the Electronic Division in the 2019 season and thanks to my job, as one of each of the 30 members of the team, we achieved the 1st place in the Italian event at Varano de Melegari.

2.2 Rules analysis

The analysis of the rules[1] is the beginning for project development: the prototype has been build in 2019 and rules belonging to that period will be presented and briefly commented. Since restrictions are related to different areas of interest this document will debates only for what concern the electric and electronic system. Voltage levels must be carefully chosen in order to supply correctly the vehicle; two power supply systems are present on board. A LVS is defined according T11.1 and the voltage must be chosen in a proper way to be sure not to exceed any limits. At the beginning no constraints in terms of voltage are given but at the end the minimum voltage necessary to operate any module (ECU, VMU or Inverters) will be considered the lowest LV voltage possible. A TS voltage is defined according to EV4.1 and the maximum permitted is 600v. Two choices can be made at this step:

- One TS accumulator plus DC-to-DC converter to supply LVS
- One TS accumulator plus one LV battery with/without regulator

The overall energy must be stacked in one or two different accumulators, the first solution in any case entail a small supply battery to be able to ignite the vehicle.

This battery must be carefully handled and connected to the low voltage system (the low voltage is provided via an isolated DC-to-DC) and a management unit such as BMS must be used according to T11.7.7 (In this document chemistry different from lithium-poly will not be taken into account due to the highly energy density of this technology). The second solutions seems more straightforward and the problem is reduced to simple sizing two accumulators without any regulator according to the longest (in terms of time) event. Each subsystem will be provided by its own regulator with the highest acceptable voltage range. The high voltage accumulator must be managed by proper electronics such as low voltage accumulator. The architecture of the BMS for the high voltage accumulator in this document will not be discussed since provided by a sponsor[1] but the interface and the main concept will be explained in the following. LV system must supply all the secondary services needed to the correct behavior of the vehicle such as sensors, data acquisition units, VMU and inverters. Once dimensioned the power supply for both tractive system and low voltage system, rules focuses their attention on safety devices. At least two safety devices must be settled in the vehicle: according to T11.6 and EV4.10. The former is the BSPD and the latter is the TSAL. Both system must be hardwired (T11.6.1 and EV4.10.8). The peculiar feature these devices must implement is something not commercial or available in the market hence custom solution have to be provided. The BSPD must be a stand alone device with non-programmable electronics, must shutdown the vehicle under certain circumstances such as combination of power delivered to motors and hard breaking condition. This device as specified in T11.6.4 must be enclosed into a single printed circuit board without any additional functionalities. TSAL is a light indicator of the high voltage relay status that must have a specific behavior according to EV4.10.2 and EV4.10.3. These conditions implies measurement on the high voltage bus in two different nodes[2] (with perspicacity of galvanic isolation) and the lighting of a light that have to be placed according to EV4.10.6. Hardwire electronics must be exploited without any possibility of using software control. These safety units interrupts the shutdown circuit described in EV6.1 and analyzed in details in the followings and a error latching unit must be implemented according to EV6.1.6. The tractive system must be activated and deactivated by the driver according to EV4.11 and a minimum driver interface must be provided, buttons and light indicator have to be place in the cockpit. At least three indicator must be placed in visible position, even in bright sunlight in the cockpit according to EV5.8.8 for the BMS, EV6.3.7 for the IMD and EV4.10.9 for the TSAL. EV4.11 specify the procedure the drive must be able to perform to activate and deactivate the TS and to set the vehicle in ready-to-drive mode. Once entered in this state the vehicle must perform a characteristic sound according to

[1] Podium Engineering

[2] Across DC link capacitors and at the vehicle-side of the accumulator container

EV4.12. Thanks to this brief background a set of mandatory requirements takes shape.

2.3 Previous year Vehicle

During 2018 season a vehicle has been designed and manufactured, the SC18. After one year of tests and debug the vehicle travels about 1000 Km and some critical points arises from the electronic architecture. A block diagram is presented below in figure 2.1.

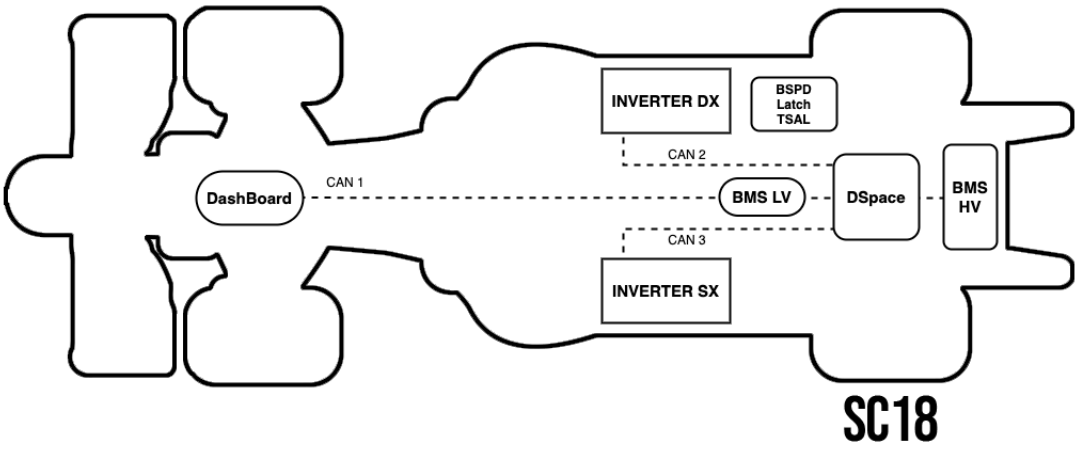

Figure 2.1: Electronic architecture of SC18

As shown above many control units has been used, each one of them performing its service. The main structure is composed by a VMU connected through a CAN bus line with four inverters, each one per wheel. The high voltage accumulator has its own BMS connected to the VMU via CAN and a low voltage battery with its own BMS has been used. Security functionalities such as TSAL and BSPD are performed by dedicated units and a single unit called DashBoard manages both cockpit and pedal signals. A the end of the '18 season a list of improvements has been released as follow:

- **DashBoard**: a microcontroller unit with no more than seven analog input was used with its internal 10bit ADC. Signals were not conditioned, buffered, clamped or filtered before been acquired hence no protection against voltage spikes or noise filtering were performed. The unit also performed cockpit management hence drives LEDs.

- **BMS LV**: a battery management system for the low voltage accumulator was used due to rules. A commercial solution has been chosen due to the simple integration into the system, but the lack of documentations creates problems.

- **Debug**: it was impossible to perform diagnostics in vehicle. Each unit had to be unmounted and placed on the bench in order to perform any type of debug. Some units that must be composed by hardwired electronics had no predisposition for debug even on the bench.

- **Update**: if a firmware update on custom units has to be performed, each unit have to be connected to a proper programmer and disconnected from the vehicle.

- **Connectors**: each unit was not designed to have a pcb-mount connector. Each unit have its own enclosure and the connection with wiring harness was made by a first connector soldered on the pcb thought wires bringing signals to a panel mount connector fixed in the enclosure.

The DashBoard performed data acquisition and were able to read analog signals with range up to 5V. During tests a sensor swap may happen, but since the unit was not properly designed for hot plug some channels became faulty. The resolution of the acquisition was not enough for the type of signals used and could be computed as in equation 2.1:

$$V_q = \frac{V_{FR}}{2^{N_b}} = \frac{5}{2^{10}} = 4.882mV \tag{2.1}$$

V_q is the quantization voltage, V_{FR} is the full range voltage and N_b is the number of bits used for the quantization. However signals coming from sensors with higher resolution such as *DHAB S-145*[3] could not be acquired maintaining the resolution. Cockpit management means switching on and off some light indicator positioned in the cockpit and reading the status of a switch used to power up the vehicle. The BMS for the low voltage system was a commercial unit by *OrionBMS*[4] that suffers from lack of thermistor inputs and documentation that makes the integration harder than expected. Debug is also important especially in a prototype car but control units had only few LEDs soldered on the PCB. Hence to understand the status of each unit was necessary look at the board. The last critical part refers to connectors:

each PCB have a pcb-mount connector, then a set of wires connect the pcb to a second connector places on the side of the enclosure. This solution increase the equivalent length of the wires and add some resistance (due to the contacts of the two connectors) and the time needed to crimp the whole wiring hardness since is doubled the number of pins.
A consideration in terms of *time* must be made: the design of the vehicle took place every year and the time needed to complete the assembling is without error margin. No mistakes can be made otherwise a delay is added. Once the vehicle is complete a set of race-test must be made to validate the choices made in the design phase and to adjust and customize some parameters to increase performances. Hence a delay in the manufacturing phase shifts and reduce the amount of time available for testing. This restriction involves that fundamental changes cannot be made and year-by-year just small improvements can be implemented.

2.4 Specifications

Starting from the assumptions above the current architecture will be extended and not radically modified maintaining compatibility and adding some new feature that in any case will not compromise the correct behavior of the vehicle. Here below in figure 2.2 a diagram with the proposal architecture and some comments will follow:

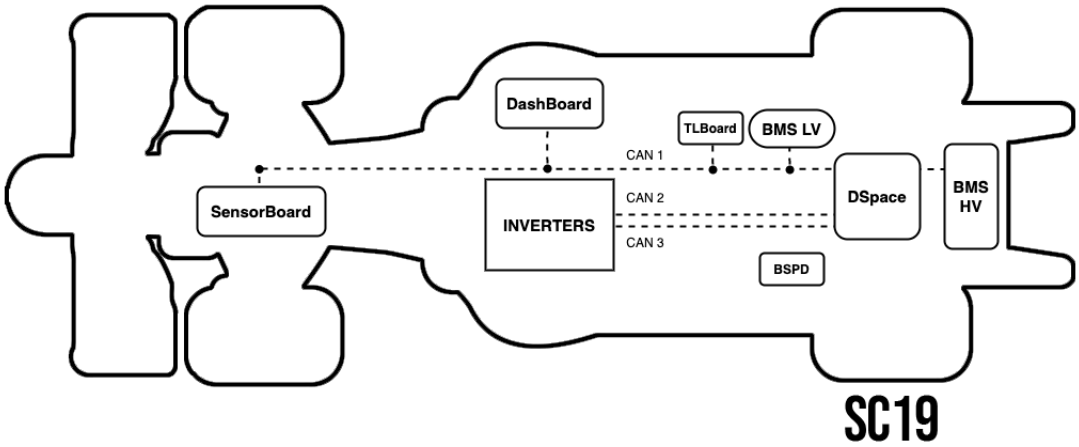

Figure 2.2: Electronic architecture of SC19

- **SensorBoard**: sample, filter analog pedal signals and send data via CAN with up to 16 inputs. The board must use a single connector to the wiring harness.
- **DashBoard**: read vehicle status from CAN bus and switch on or off certain indicator light performing a minimal driver interface. Verify cockpit button status and send ignition command to the VMU.
- **TLBoard**: integrate TSAL functionalities, AMS and IMD error latching into a single board reducing dimension and weight of the system but maintaining both subcircuit separated. The circuit that performs high voltage detection will be redesigned and distributed inside both high voltage accumulator and inverter. The unit will be equipped with a microcontroller for debug purpose.

- **BSPD**: problems relating to current sensor will be fixed and the unit will remain a stand alone device due to rules reducing as much as possible the number of pins in the connector and providing outside an analog error signal.
- **BMS LV**: a small battery management system will be designed performing current measurement, single cell voltage measurement, cell balancing, OV protection, UV protection, and overcurrent protection.

Units equipped with a microcontroller will be capable to upgrade the firmware directly in vehicle removing needs to unmount the board because of the use of an open source Bootloader. This ecosystem will work in cohesion connected to a custom windows tool able to perform debug and status monitoring in order to speed up development process.

Chapter 3

Hardware

The chapter presented below will treat the hardware development of the electronic architecture. The initial idea was to implement a modular system with a repeatable approach. Hence a microcontroller has been chosen taking into account the team's previous experience with the development environment. Each unit will have the same schematic for the MCU block with the same component in order to have at the end the same circuit replicated in different units. This approach leads to a dual result: if a bug is discovered, the solution fixes all the other circuits on the other hand, the bug is replicated hence potentially affects the whole system if not discovered. The MCU will be provided by STMicroelectronics.

3.1 Microcontroller

The Microcontroller chosen is the STM32F303VCT6 due to the availability of many development board that speed up the software development process. This solution is flexible and configurable on needs. Below in figure 3.1 the block diagram of the microcontroller from the original datasheet[5].

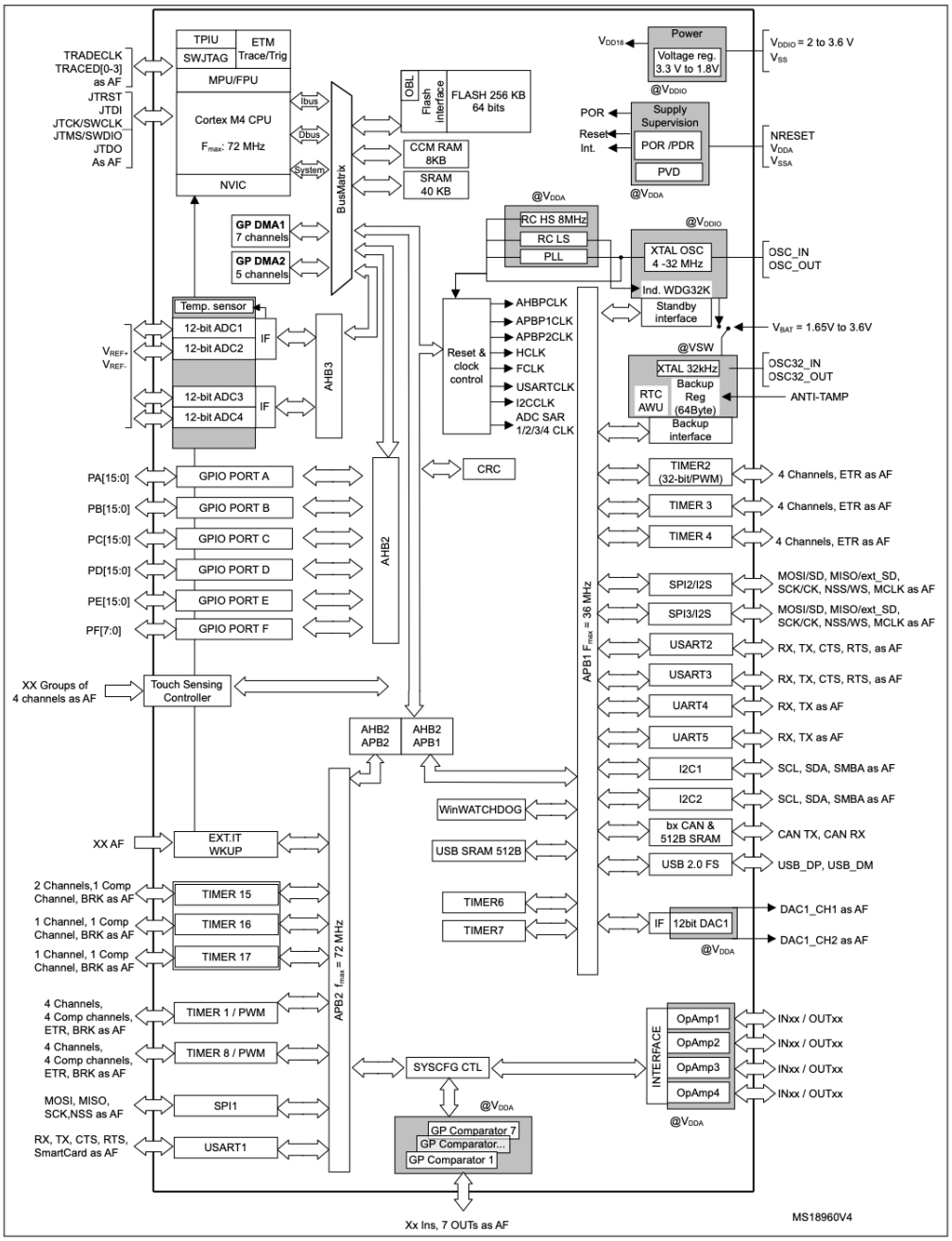

Figure 3.1: STM32F303 Block diagram

The core is wrapped with several peripherals that makes the final product compact and full of possible interface. The flash memory is up to 128 KBytes, RAM deep 32 KBytes, the number of ADC channels operating at 12-bit is up to 39, one CAN bus available, up to 31 PWM channels with a core operating frequency up

to 72 MHz. This product meets the requirements of all the units that have to be designed. The core is powered up by the power supply composed of a DC-DC converter that lower the supply voltage from 24v to 5v, and a bank of capacitors is placed in order to stabilize the voltage locally beside the core as shown in figure 3.2.

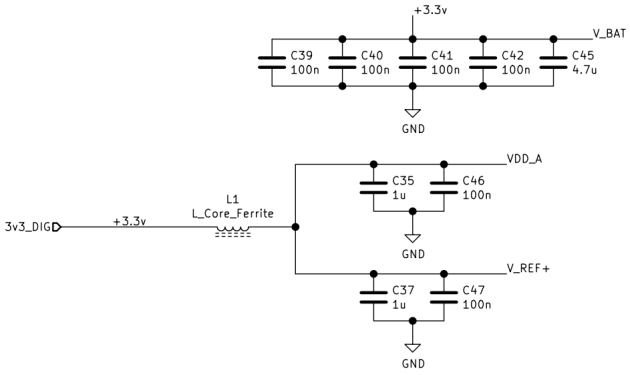

Figure 3.2: MCU Power Supply

In figure 3.3 is depicted the schematic of a generic MCU placed inside a unit: this solution is replicated in every unit taking care of different configurations and pinouts.

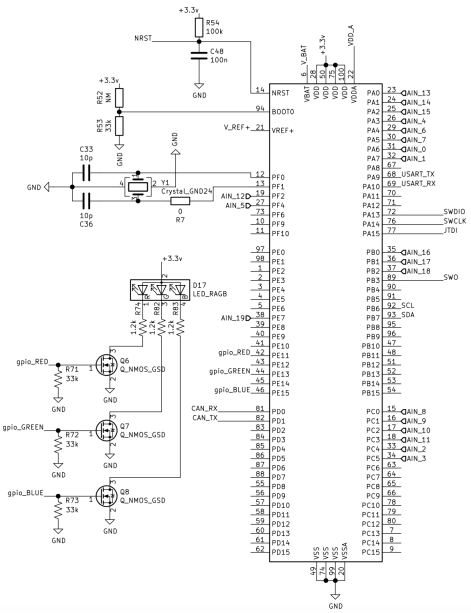

Figure 3.3: MCU Schematic

The oscillator is placed as near as possible and grounded with two capacitors as specified in the datasheet. A debug RGB led is connected via three mosfet and the transceiver CAN is provided. Particular care must be taken to the design of the CAN filtering that has been driven by guidelines provided by TI[6], in figure 3.4 the particular of the circuit.

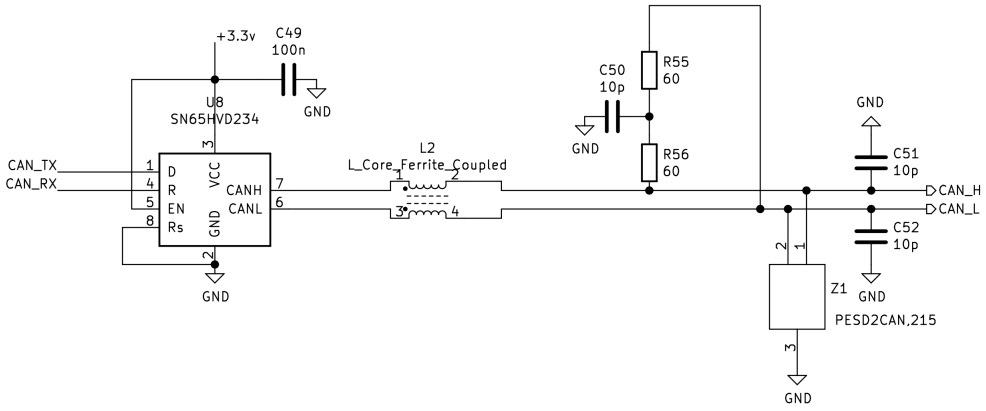

Figure 3.4: CAN Bus filter

The line is filtered with a common mode choke, and the termination of the line is set up even if not installed in every unit. The termination is also connected to a capacitor that performs filtering on the line. Bypass capacitor are also added to additional filtering and protection diode specific designed for CAN bus lines are fitted. Additional information on CAN Bus line and protocol-layer are provided in Appendix A.

3.2 Acquisition

In order to correctly acquire signals coming from pedal sensors as brake and throttle a dedicated unit has been developed, the SensorBoard due to the critical target. Signals coming from the front of the vehicle as been concentrated into this unit hence not only pedals but also steering, damper and brake oil pressure. To reduce the cost of the unit and the overall complexity the internal microcontroller ADC has been used. The main problem was the compatibility between maximum signal voltage and the maximum permitted V_{FR}; in fact the microcontroller is able to handle signals up to 3.3v. Sensors used in vehicle have an higher signal slope up

to 5v. A voltage divider has been used in order to scale the signal and lower it back to the maximum permitted value. In figure 3.5 the schematic for the analog conditioning circuitry is reported.

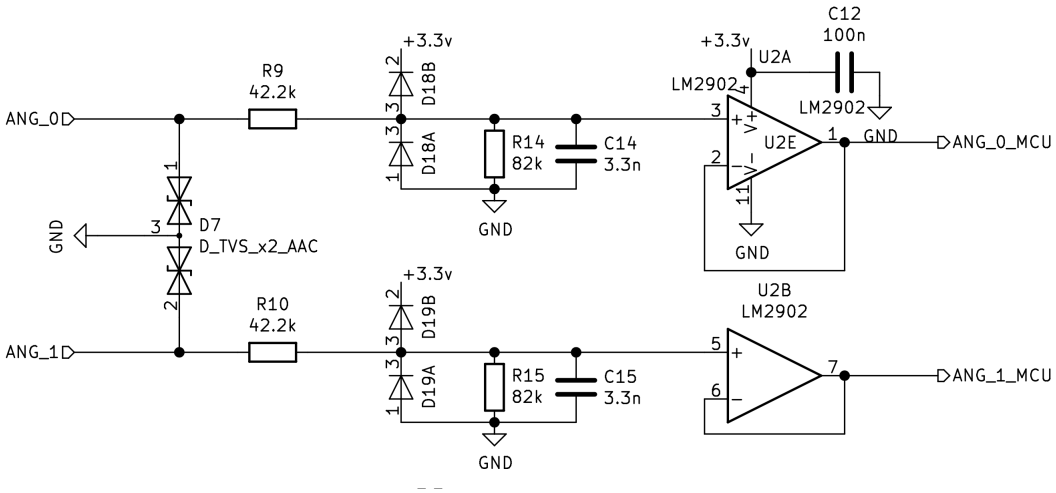

Figure 3.5: Acquisition scheme

The overall resolution is higher than previous vehicle, as shown in formula 3.1 due to the higher ADC resolution that have 12-bit dept even if the V_{FR} is lower.

$$V_q = \frac{V_{FR}}{2^{N_b} * K} = \frac{3.3}{2^{12} * 0.66} = 1.22mV \tag{3.1}$$

V_q is the quantization voltage, V_{FR} is the full range voltage, N_b is the number of bits used for the quantization and K is the coefficient introduced by the voltage divider. With this configuration specifications are met ensuring the capability of reading signals with higher resolution. Protecting sensitive units against transients and overvoltages is mandatory, hence additional components are needed. The main protection is performed by clamping diodes ensuring the signal is not allowed to reach values above the maximum permitted and below the minimum, in this scenario the maximum is the supply voltage and the minimum is the ground reference. Additional protection is assured by transient voltage suppressor diodes (TVS) placed in parallel to load they are protecting. During normal operation these diodes are transparent to the function of the circuit and interacts only if an unwanted behavior occurs. Both clamping diodes are reversed biased and if overvoltage or undervoltage occurs are forward biased allowing electric current through it. The series resistor R9 and R10 limit the current and have to be chosen in order to set the maximum tolerable

voltage at the input that is higher than the supply voltage. Here below the formula to calculate the value of these resistors in function of the current:

$$R > \frac{V_{OVMAX} - V_D}{I_{DMAX}} \tag{3.2}$$

$$R < \frac{\Delta V}{I_{OMAX}} \tag{3.3}$$

The equation 3.2 considers the maximum voltage drop across the resistor due to overvoltages divided by the maximum diode forward current. The equation 3.3 take into account the maximum current permitted by the input channel. Hence ranges of values have to be chosen in order to met protection requirements. TVS diodes respond faster than many other classes of transient suppression devices. During normal operation, a TVS diode is invisible to the circuit. In the presence of a transient voltage, they clamp the voltage across the protected load to a given level without being damaged. The avalanche breakdown effect makes this possible whereby the diode which previously was not conducting electricity begins to conduct due to the spike in voltage. The voltage divider is composed by the two resistors and the values has been chosen having a ratio of *0.66* hence to lower the maximum 5v to 3.3v The last section of the conditioning circuitry is composed by an operational amplifier in voltage follower configuration that decouple the sensor form the acquisition system, providing high input impedance.

3.3 BMS for low voltage batteries

The battery management system for the low voltage system used in the vehicle has been designed starting from requirements focusing on mechanical dimension keeping as small as possible. Below a briefly overview of what a BMS is and the type of configuration possible following the block diagram of the units designed with focus on the main feature regarding the hardware configuration.

3.3.1 Battery Management System Theory

In order to manage correctly an accumulator container a specific control unit is necessary and safety functionalities have to be implemented for rules[1] compliance. The accumulator is composed by a set of cells connected in series and then in parallel as shown in figure 3.6. The Battery Management Systems is in charged to monitoring the status of each cell: voltages and temperatures must be acquired and then processed to be sure safety conditions are verified. Voltages of each cell must be controlled but rules[1] imposes only 30% of the temperature to be measured.

Generally two configurations are available: in figure 3.6 and 3.7 are shown the differences.

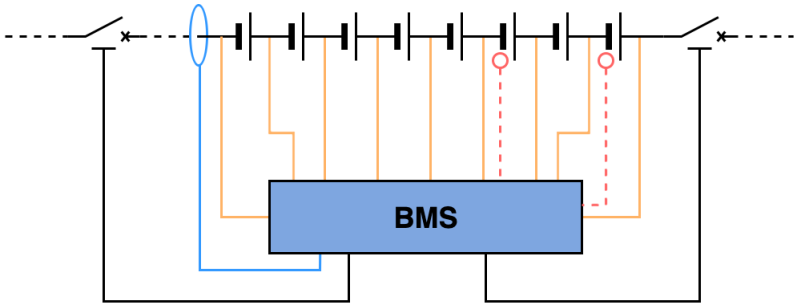

Figure 3.6: Centralized BMS

The centralized solution is composed of a single board directly connected to each cell. This configuration is not as flexible as distributed one, in fact the maximum number of cells is fixed and no further expansion is possible. The BMS also measure cells temperature, in figure 3.6 two sensors are placed as example; also the total output current is measured with a sensor.

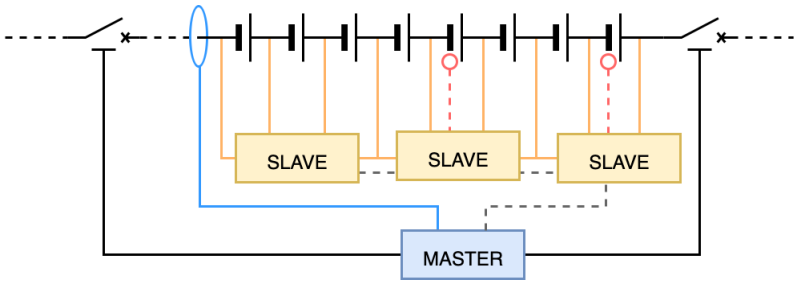

Figure 3.7: Distributed BMS

Distributed solution as in figure 3.7 have a master board that controls different slaves. Each slave performs autonomously measures as temperature and voltages and the connections with the master is performed with a daisy chain. This solution create a flexible system making future changes possible as total number of cells.

3.3.2 Battery Management System Overview

In order to develop a custom battery management system a deep analysis on the market has been performed and different solutions has been found. The main idea is to use a Co-Processor in conjunction with a Microcontroller. The former is in charge of performing acquisition on cells and the latter to read data from co-processor, and following some logic acts on consequences. Here below in figure 3.8 is reported the block diagram of the BMS.

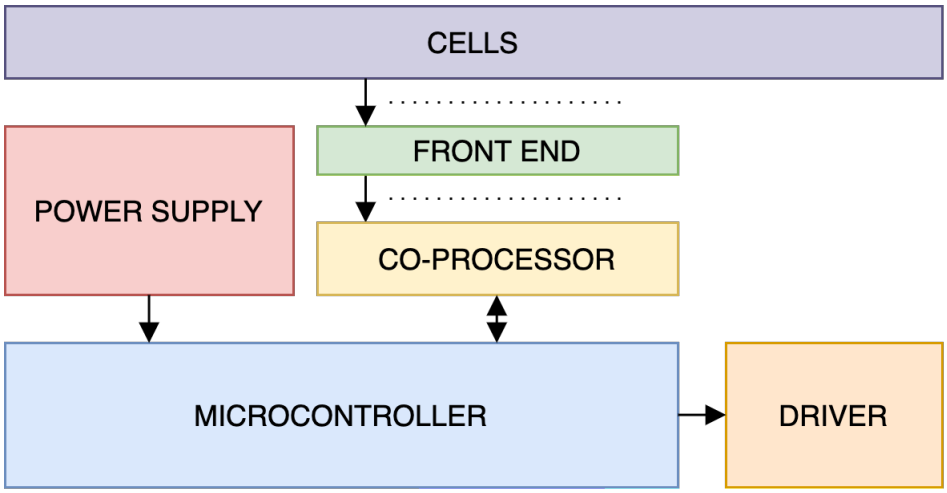

Figure 3.8: BMS low voltage architecture

The market analysis found a co-processor from TI, the BQ76PL455A[7] able to monitor up to sixteen cells autonomously with a 14-bit ADC resolution. The interface with a microcontroller host is performed via UART, in figure 3.9 the details of the component from the datasheet.

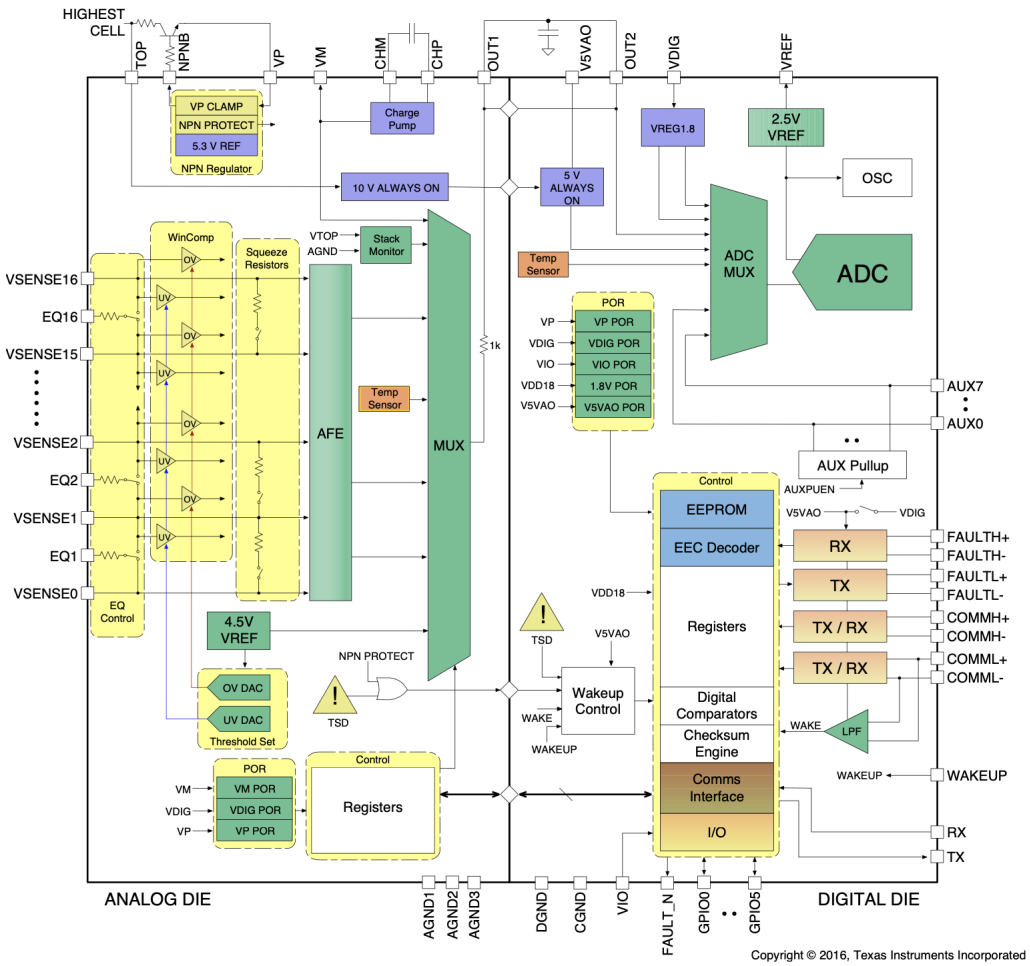

Figure 3.9: BQ76PL455A block scheme from datasheet[7]

This co-processor embedded in the same package two die, the first for the analog domain and the second for digital making this solution compact. Equipped with up to sixteen voltage sense inputs, balancing circuitry output, up to eight temperature sense inputs, self-supplied from cells, with daisy chain capability. Capable of monitoring autonomously cells and detect different faults as overvoltage, undervoltage, overtemperature and communication errors. The integrated high-speed, differential, isolated communications interface allows up to sixteen BQ76PL455A devices to communicate with a host via a single UART interface as in figure 3.10.

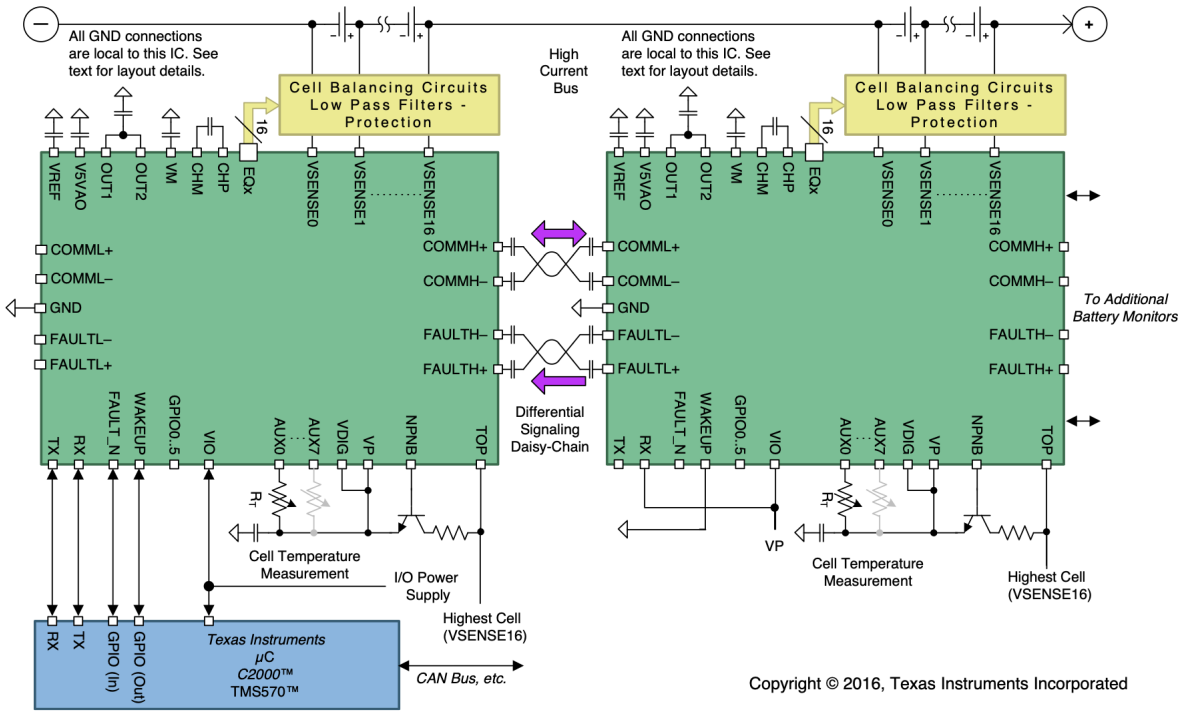

Figure 3.10: Daisy chain connection with two or more BQ76PL455A from datasheet[7]

This connection is useful in case of higher number of cells since the host remains one, hence the circuit is simply replicated identical. Is also feasible to implement a distributed solution where slave integrates only the co-processor. In order to perform balancing of cells and voltage sense few components must be included and connected as mentioned in the datasheet, here below in figure 3.11 is reported the main schematic with the interconnections.

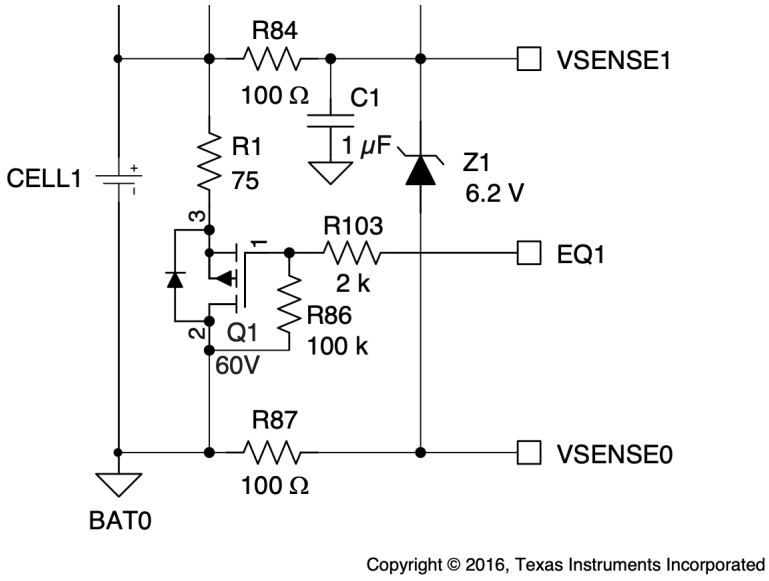

Figure 3.11: Balancing circuit from datasheet[7]

The front end circuit for voltage sense is composed by an RC filter and a zener diode against reverse voltage. For what concern the balancing circuit a mosfet as a switch is used and the current flows into a 75 Ω resistor. This resistor must be carefully dimensioned and all the order surrounding components due to the heat dissipation. Also the layout of traces is critical in this section due to the high balancing currents. This architecture decouple the problem of reading each cell independently with an isolated operational amplifier, hence simplify the whole system. The Battery Management System for the High Voltage accumulator use the same chipset with a distributed architecture hence slave pcb with BQ76PL455A and passive only are available. The pcb slave is only composed by the co-processor and passives components with two Independent connectors, the former for voltage and the latter for temperature sense. One connector is used for the UART communication with the microcontroller. Since the BMS due to specification have to be capable of reading the current provided by the battery a current sensor is mounted and a relay driver is placed in order to disconnect or connect the battery. In figure 3.12 the schematic of the current sensor's connections and in figure 3.13 for the low side driving of the relay.

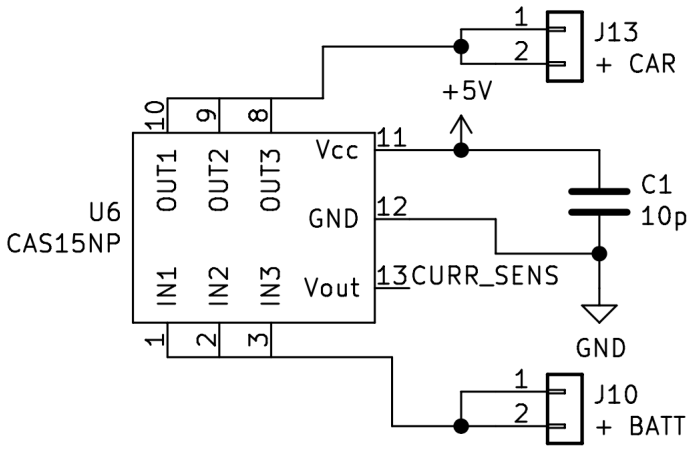

Figure 3.12: Current sensor connection

The current sensor is from *LEM - CAS15NP*[8] and as mentioned in the datasheet must be connected to few components as capacitor to locally stabilize the power supply. The output value have a slope of maximum 5v biased of 2,5v in order to represent both positive and negative current values. The master have the same acquisition circuitry with the same equivalent resolution mentioned in the acquisition chapter with the same physical low pass filter pole, then the signal will be filtered digitally.

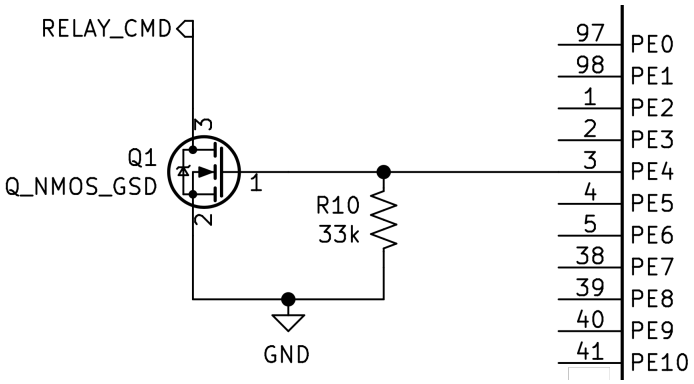

Figure 3.13: Relay low side driver

The relay disconnect the whole battery to the vehicle hence a particular scheme has been used in order to perform a supply chain. As in figure 3.14 the positive pole of the battery is connected to a mechanical switch (A) positioned on the vehicle's side. Once closed the BMS is power on and is able to perform diagnostic on cells, if

a safe state is met the BMS close the relay (B) the supply the vehicle. The supply chain is reported in figure 3.14.

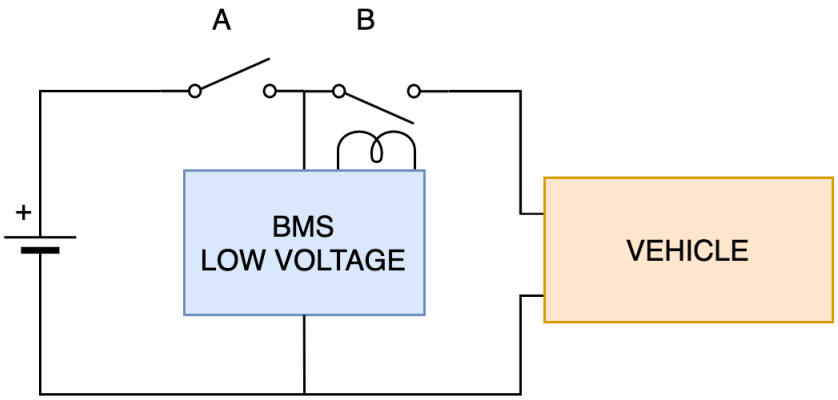

Figure 3.14: Connection to the vehicle of low voltage battery

This supply configuration is versatile due to the fact the BMS is the last in the chain and drives directly the supply relay. Additional functionalities can be implemented via software, the positive pole of the battery upstream the relay via connector is available on the vehicle side. Hence the battery can be charged directly inside the vehicle: the BMS reads the current direction and close or open the relay in order to supply the vehicle or recharge the battery.

3.4 Safety

This section will explain the main feature implemented by the safety system and begins with a briefly overview of what concern electrical vehicle precharge and discharge circuit, explaining what the shutdown circuit is. After the introduction the hardware will be outlined focusing on the high voltage detection circuit split in accumulator container version and inverter housing version then collection unit then the *TLBoard* will be described.

3.4.1 Precharghe and Discharghe

This section will explain the key points of an Electric Vehicle (EV) high voltage wiring diagram, starting from the accumulator[1] connection to inverters[2], from the safety functionalities that must be implemented. Usually an electrical vehicles have

[1]The accumulator is a set of cells connected together to store and delivery energy, is the main energy storage of an electric vehicle

[2]The inverter is the motor driver capable to convert direct current into alternating

an accumulator directly connected to a DC link capacitance as shown in figure 3.15. The capacitance simulate the inverter's input load.

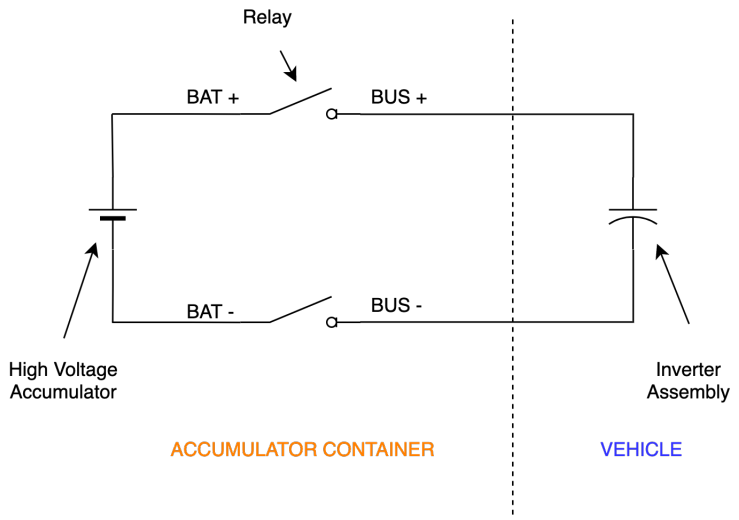

Figure 3.15: Simplified accumulator to vehicle connection

Two high voltage relay[3] are used, the first in the positive terminal and the second in the negative in order to electrically disconnect the accumulator from the vehicle. Nodes are named *BAT* if the connection is made directly in the battery terminal and *BUS* if the connection is below high voltage relays. A dedicated unit the Battery Management Unit (BMS)[4] is responsible for the startup procedure: the negative relay in closed and become the reference point for the high voltage potential and then the positive can be closed. A problem arises: the capacitance have to be charged, in a certain amount of time, before the positive relay can be closed, otherwise the current that flows will damage the capacitors since the equivalent series resistance is low. The circuit in figure 3.15 is not capable of pre-charging the capacitor bank and a modification is need adding a resistor in series as shown in figure 3.16.

[3]High voltage relay in formula SAE are known as AIRs
[4]In formula SAE usually named AMS, Accumulator Management Unit

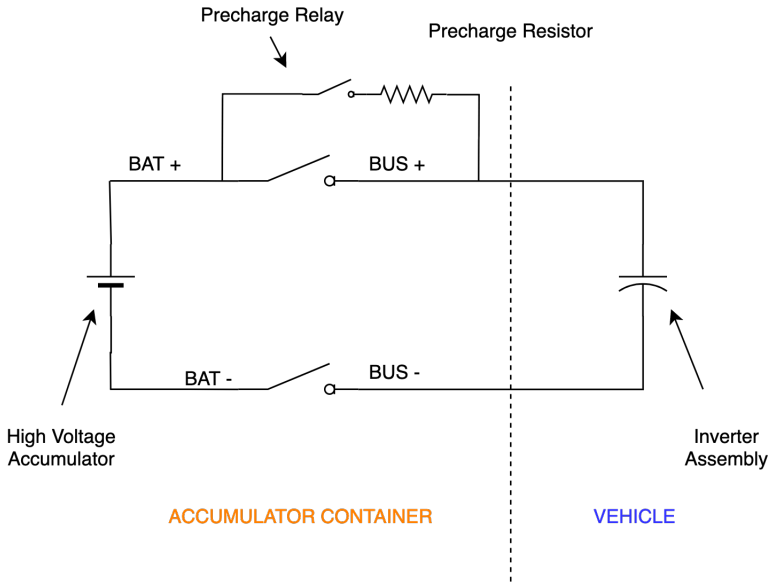

Figure 3.16: Simplified accumulator to vehicle connection with precharge

The current flowing into the capacitor follows the equation 3.4:

$$i_c = \frac{V_{batt}}{R}(1 - e^{\frac{-t}{RC}}) \tag{3.4}$$

The value of R can be chosen in order to set the time[5] needed to precharge procedure according to 3.5. Rise time from 10% to 90% is 2.2τ.

$$\tau = RC \tag{3.5}$$

The startup procedure managed by BMS becomes: close negative relay, close precharge relay and wait until the voltage on the BUS has become identical[6] with BAT side, open precharge relay and close positive relay. When the vehicle shuts down the relays are opened, and a potential dangerous situation occurs: even if the accumulator is electrically disconnected, high voltage potential difference is still present into the vehicle's circuitry. The solution is to insert a resistor in parallel to the capacitance as in figure 3.17.

[5]Generally in Automotive the precharge procedure is so fast that customer is not aware
[6]Usually a difference of 10% is negligible

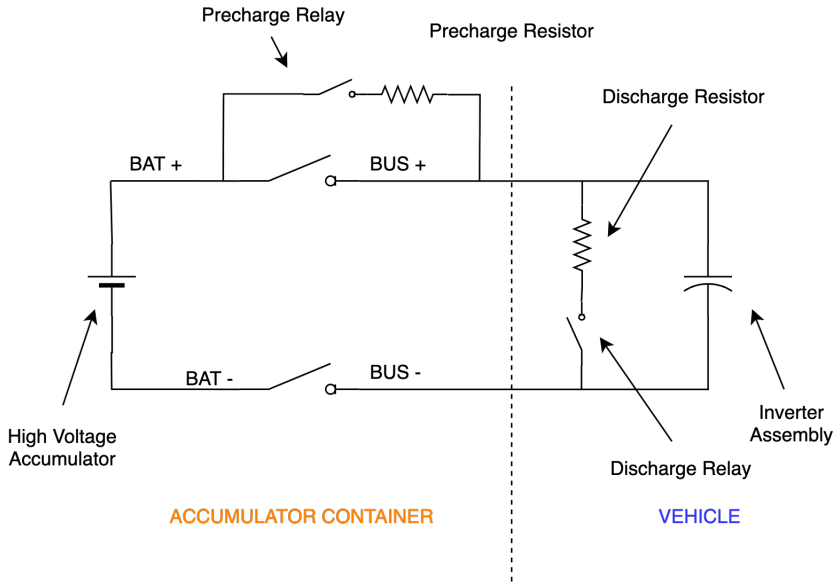

Figure 3.17: Simplified accumulator to vehicle connection with precharge and discharge

Particular attention must be taken to the position of the discharge circuitry: is placed inside the vehicle like that even disconnecting the accumulator container is possible to perform discharging. The relay used is normally closed and is supplied by the same signal that supply high voltage relay since when high voltage relays are closed, the discharge is opened and vice versa. The value of discharge resistor have the be choose in order to set the discharge time according to the following 3.5 and the power of the resistor have to be chosen according to maximum current that flows 3.6.

$$i_c = \frac{V_{batt}}{R}(e^{\frac{-t}{RC}}) \tag{3.6}$$

3.4.2 Shutdown Circuit

The shutdown circuit is a safety system of a Formula Student vehicle, is defined in rules [1] and will be described in details in this section. The basic idea is that this circuit supplies directly AIR and if ad unwanted situation occurs this circuit is interrupted hence AIR not supplied anymore. The figure 3.18 illustrate a generic block diagram of the circuit.

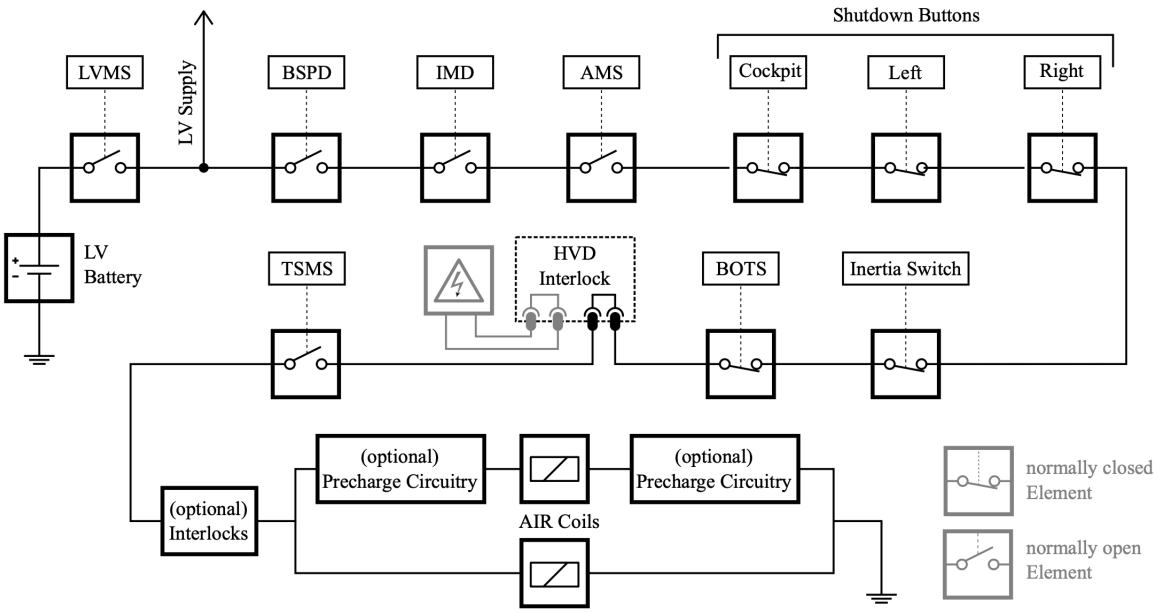

Figure 3.18: Shutdown circuit diagram from [1]

It starts from a low voltage battery and in series a set of switches or relay are connected, each one of them performing a different safety function. In vehicle are installed different safety units, in figure 3.18 are represented the mandatory ones. Starting from the left, Low Voltage Master Switch (LVMS), is a physical switch placed on the right side of the vehicle, used for switch on low voltage system. Once switched on LV the first safety device is the BSPD; it performs its safety function continuously and if safe condition is verified close a normally-opened relay. The same for IMD and AMS. The first three elements are composed by electronic devices, others are physical switches placed in the vehicle in order to be actuated by drive or people around: switches are places in the cockpit, on the left and right side of the main-hoop. These physical actuators are normally closed in order to be actuated hence open the shutdown when a critical situation occurs. Inside the vehicle is placed an inertial switch that opens the shutdown if a collision or a shock happened. After the inertia, a switch mounted into the brake pedal verify the if pedal position exceed the maximum, the BOT. The next point is the HVD, a physical connector that interrupts the high voltage path, hence have to interrupt the shutdown as rules compliance. This connector is usually removed when performing handwork on the vehicle. Last step is the TSMS, a physical switch placed next to LVMS. From that point on the shutdown could supply AIRs.

3.4.3 TLBoard

High Voltage Detection - Slave

Inside the accumulator container an high voltage detection circuit is placed and is a part of a master-slave architecture. In fact this small slave only detects if the voltage across the high voltage bus is above or below a certain threshold. The circuit is also able to detect if AIRs are individually opened or closed performing cross measure between them. In order to perform a cross measure in both downside and upside AIRs connection two isolated power supply have to be generated as in figure 3.19.

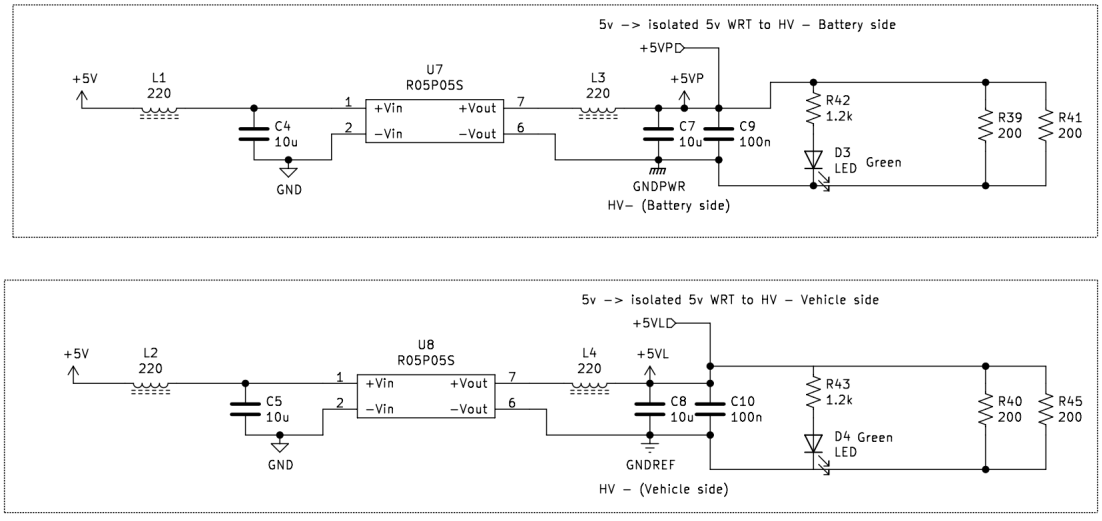

Figure 3.19: Isolated Power Supply for High Voltage detection

The power supply used requires few passives around and a minimum load composed by two high power resistors connected to the output. To perform cross measure two different potential reference are needed as in figure 3.19. The first starting from the top is connected in the battery negative pole and the second is connected to the negative pole of the vehicle side. A more detailed scheme is reported in figure 3.20 to better understand the connection.

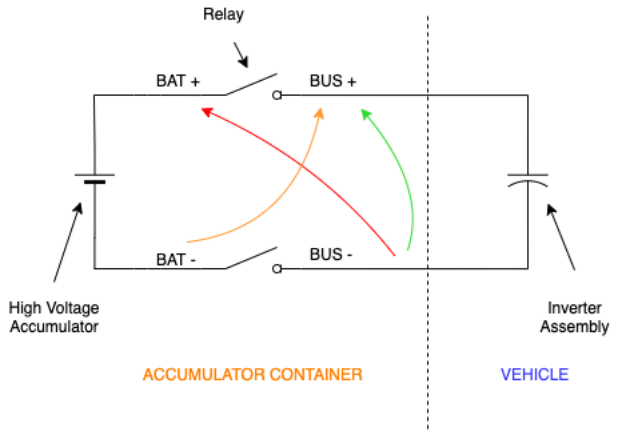

Figure 3.20: Connection for cross measure

With this scheme is possible to determine if one of the AIR is stuck at closed. In fact the measure is performed in with cross references: the voltage above the positive relay is measured with reference the battery side and with the vehicle side. Also the voltage upside the positive relay is measured with reference the vehicle side. This setup creates three signals: a negative AIR closed signal, a positive AIR closed signal and a generic High Voltage detection signal. Rules imposes to verify coherence between wanted AIRs state and the real one, also the voltage across DC link capacitors must be the same as downstream AIRs. A generic detection circuit is reported in figure 3.21 and the same is replicated three times for AIRs status detection and once inside the inverter.

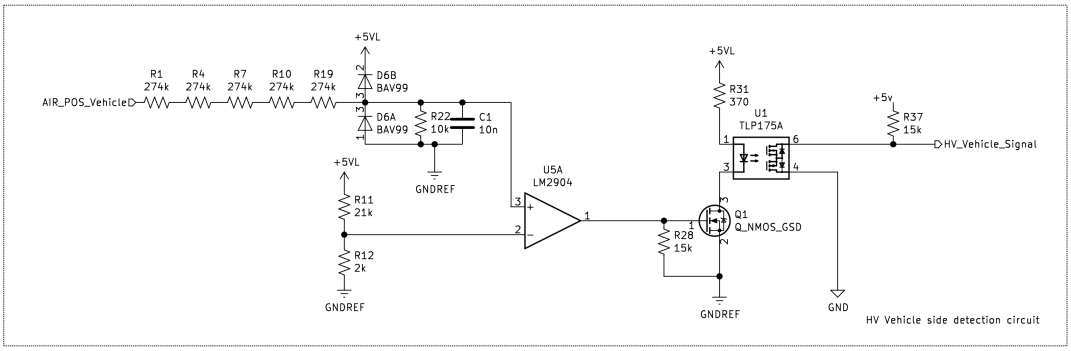

Figure 3.21: Measuring circuit for High Voltage Detection

The circuit is able to detect if the voltage is above or below a certain threshold as rules states, if the voltage across the input is higher than sixty volts the comparator is triggered switching on the mosfet which pulls down the cathode of the optocoupler that triggers its output. Protection diodes are used in order to clamp input to the

power supply to prevent overvoltages. The input voltage is first divided by a factor of 139 then compared with the output of a voltage divider that set the threshold of 0.434v corresponding to 60.3v. The same circuit is copied and used inside the inverter container in order to detect implausibility.

High Voltage Detection - Master

The *TLBoard* is composed of three subcircuits:

- **TSAL driver**: the logic that drives the TSAL.
- **Error Latching**: the latching circuit for the IMD and BMS errors.
- **MCU**: a core that wraps all the logics performing debug and monitoring of the unit.

Due to rules the TSAL light must be driven by non programmable logic hence logic gates has been used. The circuit reported in figure 3.22 driver the light and is connected with the slaves explained in 3.4.3.

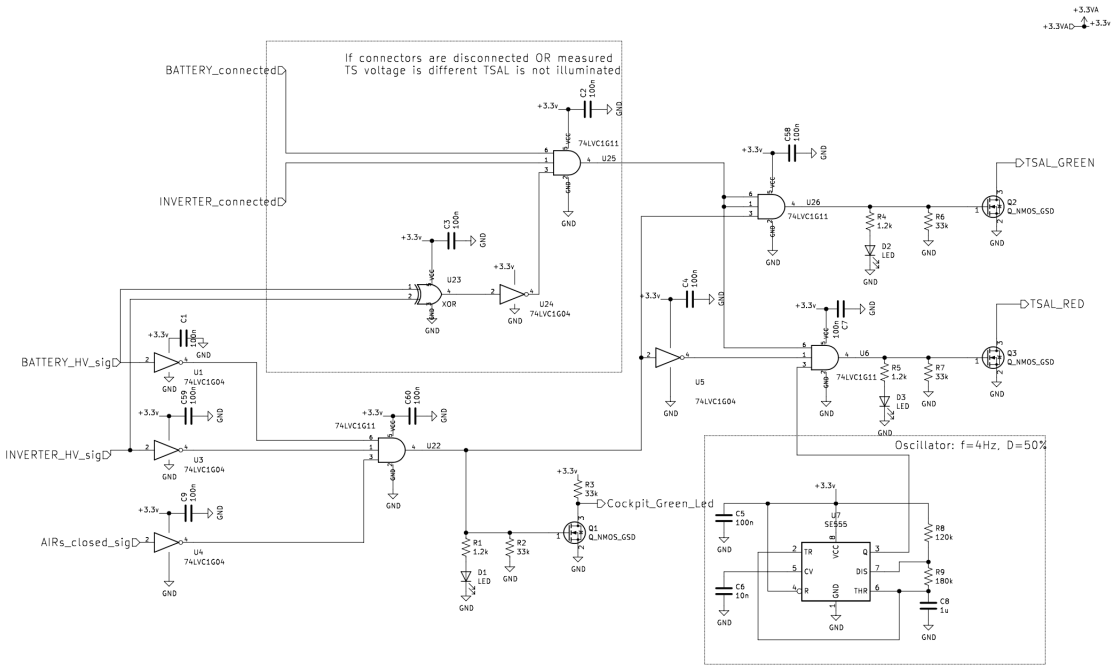

Figure 3.22: TSAL light logic driver

The input signals coming from outside the unit are connected to the logic gates via dedicated mosfets in order to guarantee electrical decoupling hence the signal logically must be inverted. Here the explanation of *NOT* gates placed in inputs. From the slaves arrive three signals, the first from the inverter that detects high voltage is present, a second coming from the battery accumulator that identify high voltage is present downstream AIRs and the third signal that indicate both AIRs are closed. [7] Other signals arrives from the outside, in fact the unit is able to detect if both battery and inverter battery are relatively connected to the vehicle. If the three signals have low logic value an *AND*[8] logic gate will result to high logic value. This logic gate drives two other *AND*[9] gates that drives the TSAL light via pulldown mosfets. The *green* light is on and stable if connectors of both battery pack and inverter are connected, and high voltage is not present. The *red* light is on and flashing with a frequency of 4Hz [10] if connectors are connected and high voltage is present in both inverter and battery pack slave. The light is off representing a implausibility condition if signals coming from slaves are different. Logic gates have output connected directly to pulldown mosfet in order to drives light without any possible fault and one signal is brought to microcontroller in order to perform debug.

Error Latching Circuit

The error latching circuit is mandatory due to rules and latch IMD and BMS errors. These errors are sent to the *TLBoard* analogically with single signals. The schematic of the circuit is reported in figure 3.22.

[7] The slave detect each AIR individually, but internally the signal is placed in *OR* condition in order to reduce the number of pins and the wire used in the wiring harness

[8] U22

[9] Respectively U26 and U6

[10] Due to the oscillator U7

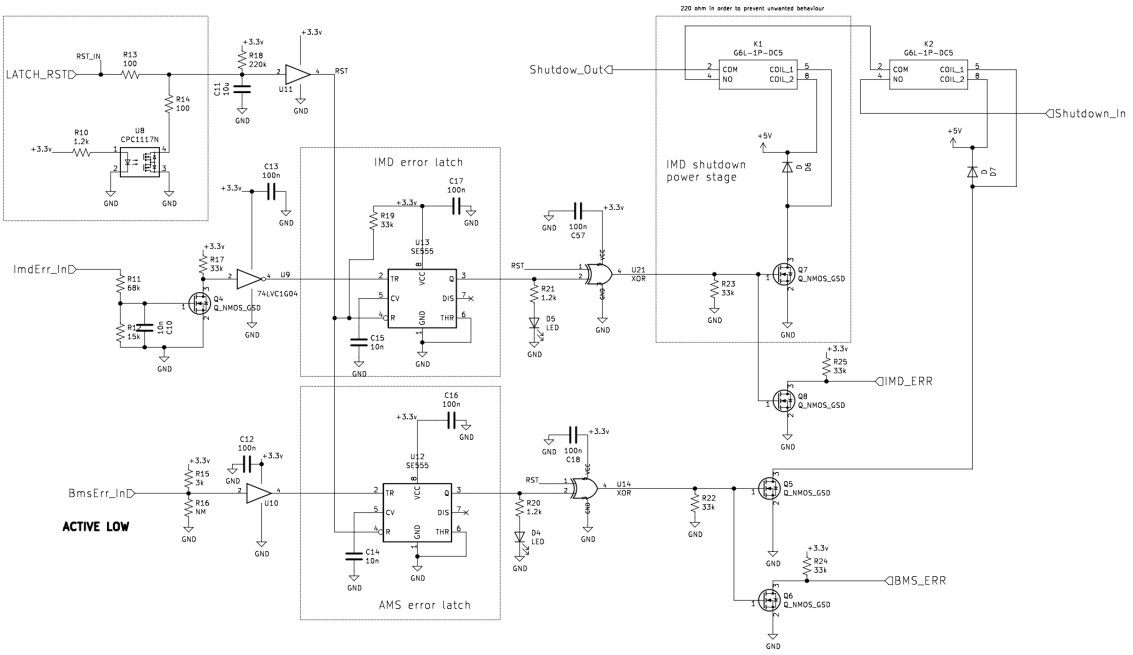

Figure 3.23: Error Latching circuit

Inputs are connected to oscillators[11] used in trigger mode. Once triggered outputs remains high inutile reset pin is pulled down. Oscillator are triggered by error signal via mosfet[12] or buffer in order to decouple the signal. The latching circuit can be reset with a pushbutton positioned on the vehicle side and that pulls down each reset pin of the oscillator. In order to maintain the error status if the pushbutton is pressed while error persists a *XOR* gate is connected to the oscillator gate with the reset. This circuit directly interrupt the shutdown circuit if the error status is reached driving two separated normally opened relays. The output status of the circuit is brought outside with two separated mosfet and read by the microcontroller for debug. This separation is needed in order to guarantee clearness in the circuit that must be approved by judges before events. Hence all non programmable logic of the *TLBoard* is monitored with dedicated output signals that are red by the microcontroller and sent via CAN. The signal from the TSAL is used then for drive a cockpit led that warn driver high voltage is present.

[11] NE555

[12] In case of IMD error due to the incompatibility of voltages level that reaches 24V

Chapter 4

Software

This chapter will treat in details the software's architecture focusing on the finite state machine developed. Since the microcontroller used is the same for multiple units also the software structure is similar. The main idea that drives the development was to replicate also the software part hence a component-based solution has been implemented. There are components common to units such as low level drivers and higher level components that are custom designed for the specific application. A specific development environment has been set up and a git repository was used to manage code revision. At beginning custom hardware solutions were not available hence the original code was developed on evaluation board and multiple board connected together via shields were set up to simulate the vehicle. Once the hardware was ready to be mounted the firmware has been customized for the final hardware revision and then installed and ready to be tested in field.

4.1 Architecture overview

The software component developed are the same for every unit. The firmware is composed of different components that are reporter in figure 4.1

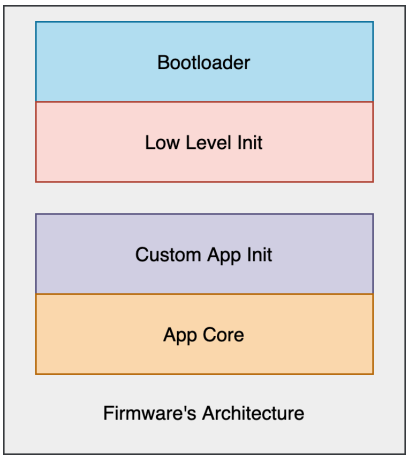

Figure 4.1: Firmware's architecture

The first module is the *Bootloader* that will be described in details in the following chapter. The second module is composed of all necessary low level functions that initialize correctly the microcontroller, since the hardware is the same for every board, this module is also common to every unit. Initialization functions and parameter had been generated automatically by a proprietary STMicroelectronics graphical tool known as *STMCubeMX*[10]. This tools permits a safe and robust initialization since verify mismatches and eventually configuration errors generating a source code file that should not be modified. Below the low level initialization module a custom application initialization module is placed in order to be able to modify the pinouts depending on application. At the end the core application module that implements specific application features.

4.2 Bootloader

The bootloader is a software module normally used in modern electronics allowing embedded system to be autonomously updated. This layer is common to every unit installed in the vehicle and use CAN as deploy interface. Due to the complexity and the critical functionalities that have to be implemented the main structure is from an open source solution[9]. This solution is flexible and libraries are available to manage directly at high level application. Here below a briefly theoretical overview of what a booloader is and how it should behave.

4.2.1 Bootloader Theory

This section will explain why is necessary the use of a bootloader into an embedded automotive system. Bootloader is a piece of code that is responsible for remotely code upgrade. Inside a vehicle are placed several VMU and most of them are located in unreachable places or disconnecting them from the vehicle is not possible. Prototypes like formula student's vehicles are subject to continuous software upgrade even on-the-track therefore a fast and easy procedure to upgrade VMU's firmware in-vehicle have to be implemented. Usually a bootloader is not different from a standard application, the only difference is the location in which is stored: the first section of the memory in order to be the first set of instruction to be executed as shown in figure 4.2.

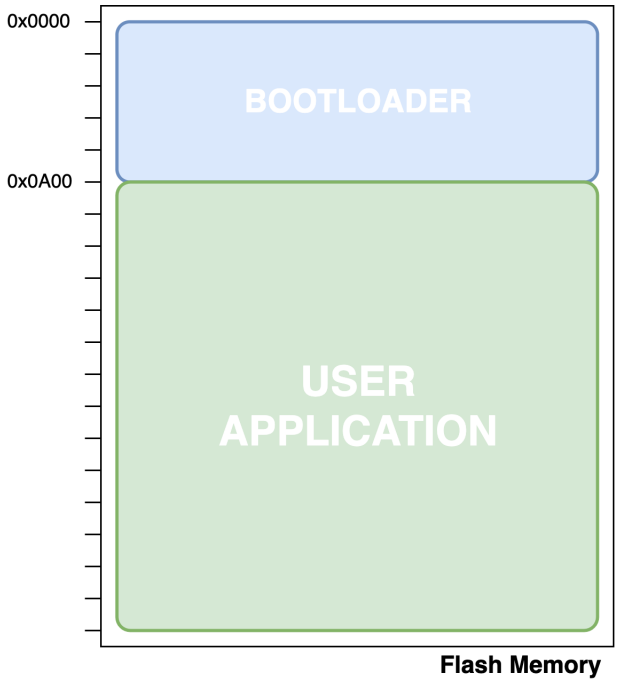

Figure 4.2: Bootloader's position

The memory is divided in two sections: bootloader and user application. Several communication protocols can be used in order to upgrade firmware, UART, CAN, TCP/IP etc. In automotive usually CAN is preferred. The VMU at boot performs the following procedure, as shown in figure 4.3.

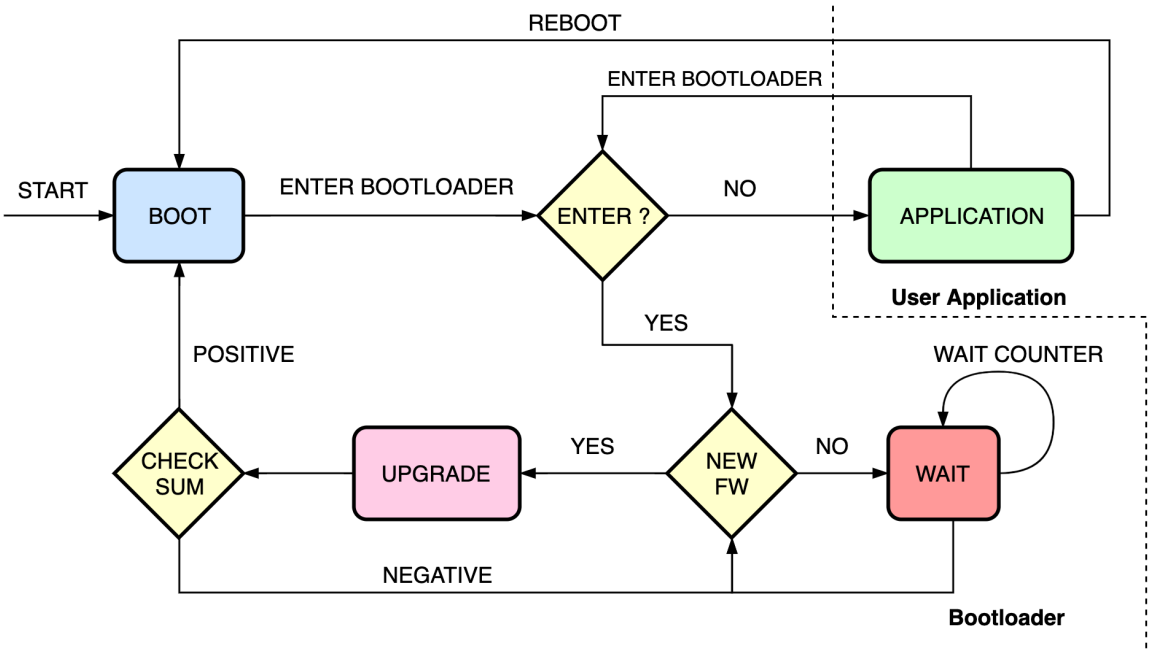

Figure 4.3: Bootloader's state machine

At power up the bootloader verify if a new firmware is available on the communication bus: if present start a procedure that erase the memory and substitute byte to byte with the new code of the user application. At the end of the communication a safety checksum is performed to be sure the firmware is coherent with the forwarded one. If checksum verification is positive the user application can be executed and bootloader terminated, otherwise the booloader remains in idle. If no firmware update is present on the bus the user application starts directly. An important feature that must be implemented is the possibility to enter in bootloader mode when the user application is running. Usually the user application is able to reboot the whole system hence force the bootloader to start.

4.2.2 OpenBLT

The bootloader core is based on the open source version but few modifications has been done in order to make is feasible for the application. The main function is listed below and is composed of few sub-functions. At beginning al peripherals are correctly set up and the internal oscillator has been used to prevent faults. During the whole bootloader task the clock is generated internally and once that task ends the clock is deinitialized to permits the user application to use the more precise external oscillator.

```
int main(void)
{
  /* initialize the microcontroller */
  Init();
  /* initialize the bootloader */
  BootInit();

  /* start the infinite program loop */
  while (1)
  {
    /* run the bootloader task */
    BootTask();
  }

  /* program should never get here */
  return 0;
} /*** end of main ***/
```

Then a infinite loop begins and wait for special commands to be received via CAN. If the command are received the bootloader is triggered and stars receiving the new firmware and installing in the previously erased flash memory. When the download is finished[1] the CRC is calculated and compared with the expected one. This make the download safe and fully reliable since the bus is not totally error-free.

[1]Special techniques are implemented for sending the firmware over the bus as XCP protocol

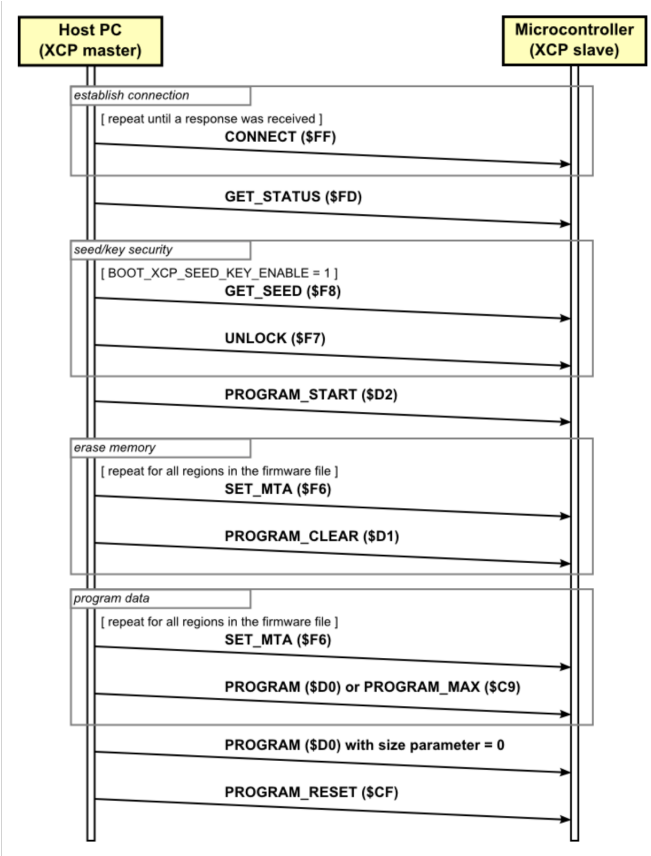

Figure 4.4: Bootloader communications with an Host

In figure 4.4 is reported the communications protocol used by an host PC that runs a custom tool that will be described in the following sections. Four state can be described:

- **HandShake**: a special command *0xFF* is send to trigger the bootloader and establish a connection.
- **Key**: a security key is fed if enabled.
- **Erease**: erase the block of memory reserved for the user applications.
- **Programm**: fill the memory with the new data provided by the host PC and calculate the CRC.

If the CRC is correct the bootloader boots the user application otherwise remains in waits for new firmware upgrade. In normal conditions, at every reset, the bootloader is not triggered hence the user application start almost immediately and

a fast check is performed at boot verifying the checksum, that is calculated and programmed at the end of a firmware update, to determine if a valid user program is present. Since multiple control units are connected to the same CAN bus the host PC must be able to address the new firmware to the correct unit with a unique ID. The special command sent to establish the bootloader connection is sent to a particular reserved CAN ID as in table 4.1 that has been chosen with the highest possible priority.

Board	ID
SensorBoard	0x001
DashBoard	0x002
TLBoard	0x003
BMS LV	0x004

Table 4.1: Bootloader CAN ID

4.3 Low Level Init

The code generated by *STMCubeMX* is composed of a main functions structured as follow:

```
int main(void)
{
        /* MCU Configuration */

        /* Reset of all peripherals, Initializes the Flash interface and the
           Systick. */
        HAL_Init();

        /* Configure the system clock */
        SystemClock_Config();

        /* Initialize all configured peripherals */
        MX_GPIO_Init();
        MX_DMA_Init();
        MX_CAN_Init();
        MX_TIM2_Init();
        MX_TIM3_Init();
        MX_ADC1_Init();
        MX_ADC3_Init();
        MX_I2C1_Init();
        MX_ADC2_Init();
        MX_USART1_UART_Init();

        /* Custom App Init */
        SetupBoard();

        /* Infinite loop */
        while (1)
```

```
28          {

30                  CoreBoard();

32          }

34  }
```

All necessary peripherals as DMA, GPIO, CAN, Timer and UART are initialized properly before the clock is configured. Then the *SetupBoard()* function is called to initialize the custom application taking into account different pinouts. After the correct initialization an infinite loop begin with the *CoreBoard()* performing the core of the application. In order to perform different tasks in a parallel-like mode a scheduler has been developed based on internal microcontroller timers: once the timer reach the defined value an interrupt is triggered. The callback function has been designed in order to increment the value of a global variable with a precise time delay. This variable is read constantly by different functions and compared with the a previous value corresponding to the last function call. If the value is greater or equal the function is executed meaning a specific amount of time has passed. Otherwise the function is not executed. This implementations permits the execution of different functions with different time delays.

4.4 SensorBoard - Acquisition Module

The SensorBoard have to acquire and filter analog sensors then send to the VMU the data; the main *Core()* is reported below.

```
  /*Core SensorBoard*/
2 void CoreSensorBoard(void)
  {
4
          SaveADCValues(2);
6         /*Read from DMA and save 10 samples x channel*/

8         FilterADCValues(50);
          /*Mean value for 10 samples each 5ms */
10
          Debug_UART(true);
12        /*Print into "value" string the content of ADC[], then write to UART1*/

14        CAN_Tx(80);
          /*Send data via CAN*/
16
          LedBlinking(LED_G_GPIO_Port, LED_G_Pin , 1000);
18        /*Blink green led*/

20 }
```

The argument passed to the function is the time delay that the function must wait before being executed expressed in hundreds of microseconds. For example

SaveADCValues(2) will be executed every 200μ seconds. The first function reads sampled values from DMA[2] and save them into a matrix with deep the number of inputs and wide a defined maximum number of samples. The code is reported below.

```
/*Fill ADC matrix with samples*/
void SaveADCValues(uint32_t delay_100us)
{
        static uint32_t delay_100us_last = 0;

        if(delay_fun(&delay_100us_last,delay_100us))
        {

                for(int j = 0; j < CH_NUMBER; j++)
                {
                        for(int i = 0; i < ADC_SAMPLE_NUMBER-1; i++)
                                {
                                        ADC_SAMPLE[j][i+1] = ADC_SAMPLE[j][i];
                                }
                        /*shift all samples by 1 position:  ADC_SAMPLE[0][1] =
                              ADC_SAMPLE[0][0]*/

                        ADC_SAMPLE[j][0]        = ADC_IN[j];
                        /*Fill first column with new samples*/
                }
        }
}
```

This procedure is useful to save the history of previous values, needed to perform digital filtering. The digital filter is implemented with moving average as follows.

```
/*Mean value for each channel*/
void FilterADCValues(uint32_t delay_100us)
{
        static uint32_t delay_100us_last = 0;

        if(delay_fun(&delay_100us_last,delay_100us))
        {
                for(int j = 0; j < CH_NUMBER; j++ )
                {
                        ADC_OUT[j] = 0;
                        /*Reset buffer*/

                        for(int i = 0; i < ADC_SAMPLE_NUMBER; i++)
                        {
                                ADC_OUT[j] += ADC_SAMPLE[j][i];
                        }

                        ADC_OUT[j] = ADC_OUT[j] / ADC_SAMPLE_NUMBER;
                }

        }

}
```

[2]The DMA writes samples from the ADC to the memory autonomously, hence the acquisition is reduced to a memory-read process.

Values saved in the matrix are summed together and the divided by the number of samples saved. The number of samples and the period when the function is called determines the cut off frequency of the low pass filter, but due to the complexity solution of the filter's equation the tuning has been done empirically taking into account also the introduced delay. Then a debug UART function is called in order to have a debug interface active; the parameter passed is used to enable or disable the interface. With this interface is possible to send debug information outside hence normally disabled during normal operation of the vehicle. The next function sends the data via CAN and is structured as follows.

```
/*Send data to CAN BUS*/
void CAN_Tx(uint32_t delay_100us)
{
        static uint32_t delay_100us_last = 0;

        if(delay_fun(&delay_100us_last,delay_100us))
        {
                TxHeader.StdId = 0x010;
                TxHeader.ExtId = 0x0010;
                TxHeader.RTR = CAN_RTR_DATA;
                TxHeader.IDE = CAN_ID_STD;
                TxHeader.DLC = 8;
                TxHeader.TransmitGlobalTime = DISABLE;

                TxData[0]= (ADC_OUT[0] >> 8);
                TxData[1]= ADC_OUT[0];

                TxData[2]= (ADC_OUT[1] >> 8);
                TxData[3]= ADC_OUT[1];

                TxData[4]= (ADC_OUT[2] >> 8);
                TxData[5]= ADC_OUT[2];

                TxData[6]= (ADC_OUT[3] >> 8);
                TxData[7]= ADC_OUT[3];

                CAN_Msg_Send(&hcan, &TxHeader, TxData, &TxMailbox, 30);

                ...
        }
}
```

The structure *TxHeader* is used to set the CAN ID and other parameters that are explained in details in AppendixA. The structure *TxData* represent the byte to be send and the function *CAN_Msg_Send()* adds the message to a queue ready to be send on the bus. The last function switch on or off intermittently a RGB LED placed on the board that is useful to understand the unit is working correctly. If the unit is booting the LED blinks blue and then if everything works correctly blinks green. If an error occurs the LED remains fixed in red.

4.5 DashBoard - Cockpit Management Module

The DashBoard unit have to read from CAN signals and switch on or off certain LEDs. Also perform a finite state machine needed to make the vehicle in ready to drive state. The main *Core()* function of the DashBoard is reported below.

```
/*Core DashBoard*/
void CoreDashBoard(void)
{

        LedBlinking(LED_G_GPIO_Port, LED_G_Pin , 1000);
        //Blink green led

        UpdateCockpitLed(5000);
        /*Update state Cockpit's LEDs*/

        ReadyToDriveFSM(500);
        /*Ready to drive FSM*/

        UpdateOnTime(20000);
        /*Update EEPROM counter value*/

        CAN_Tx();
        /*Send timer data via CAN*/

        Debug_CAN_Tx(500);
        /*Send debug packet*/

}
```

The first function perform LED management. Then *UpdateCockpitLed()* is called every 500 ms. The DashBoard receives a signal via CAN named *Error Byte* containing the status of BMS, IMD and High Voltage Bus and switch on the corresponding red LED showing if an error occurs. Due to the safe nature of this signals this task must be fail safe, hence if the *Error Byte* is not received for any reason the error status is reached. Below is reported the code.

```
/*Update Cockpit's LEDs*/
void UpdateCockpitLed(uint32_t delay_100us)
{
        static uint32_t delay_100us_last = 0;

        if(delay_fun(&delay_100us_last,delay_100us))
        {

                if(ERR_BYTE_RECEIVED)
                {
                        ERR_BYTE_RECEIVED = false;
                        HAL_GPIO_WritePin(GPIOE, BMS_LED_Pin, BMS_ERR);
                        HAL_GPIO_WritePin(GPIOE, NOHV_LED_Pin, NOHV);
                        HAL_GPIO_WritePin(GPIOE, IMD_LED_Pin, IMD_ERR);

                        /*LED ON or OFF depending on ERR_BYTE*/

                }
```

```
                else
                {
                        HAL_GPIO_WritePin(GPIOE, BMS_LED_Pin, ON);
                        HAL_GPIO_WritePin(GPIOE, NOHV_LED_Pin, ON);
                        HAL_GPIO_WritePin(GPIOE, IMD_LED_Pin, ON);
                        /*LED always ON, timeout can*/
                }

        }

}
```

If the signal is received an interrupt is triggered and the status of corresponding LED is saved into global variables. *ERR_BYTE_RECEIVED* is a global variable that is *true* only if the signal is received and false vice-versa. This ensure the fail safe condition. The most complicated part is the finite state machine that have to read the status of a push button in order to be triggered. Once started a set of particular commands are sent to the VMU. The communication mechanism implemented follows the following structure: the DashBoard send a command to the VMU and wait for an aknowledge. If the aknowledge is received the state changes otherwise remains in wait until a timeout expires. The finite state machine graph is reported in figure 4.5.

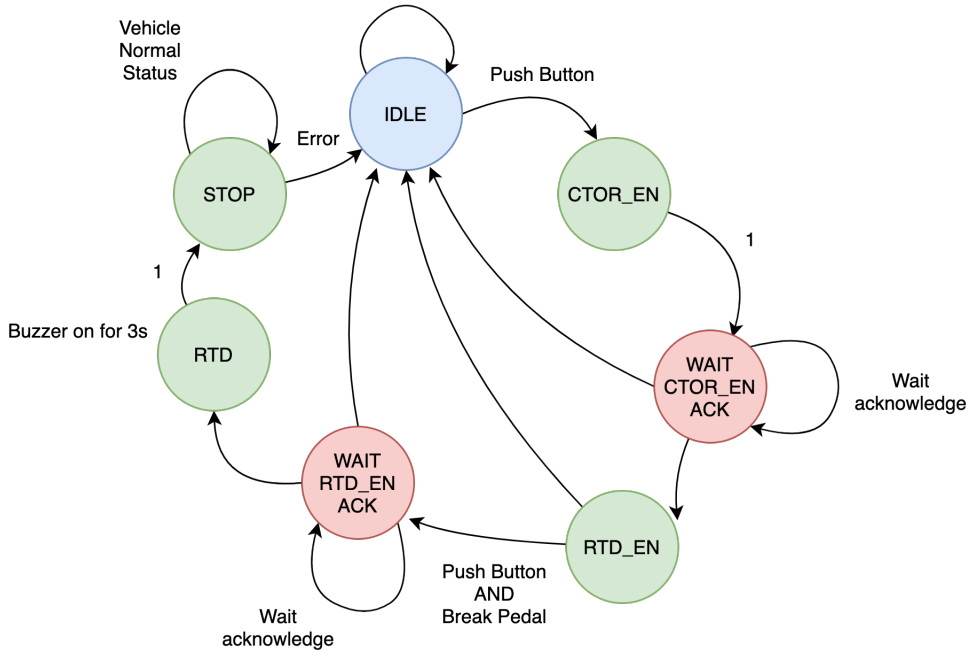

Figure 4.5: Finite State Machine to ignite the vehicle

The finite state machine starts in *IDLE* and waits to be triggered by the push button positioned on the cockpit. When pushed changes state into *CTOR_EN* and

send to the VMU a command to close AIRs. Then when the message is sent goes into *WAIT_CTOR_EN_ACK* waiting for the aknowledge. If received goes into *RTD_EN*. The vehicle in this state is almost ready, AIRs are closed and High Voltage is present on the bus. Due to rules the ready to drive condition have to be entered by simultaneously pushing the cockpit button and the brake pedal. If this condition occurs a message is sent to the VMU and the state goes into *WAIT_RTD_EN_ACK* waiting for the aknowledge. If the aknowledge is received an acoustic emitter is switched on for few seconds due to rules and a LED on the cockpit is switched on to warn the driver the vehicle is ready to drive. The VMU is able to reboot in any conditions the state machine sending a particular command hence if the vehicle is switched off there is no need to do a power cycle.[3] The unit is equipped with an EEEPROM in order to store data. For analysis the amount of time elapsed in ready to drive state is saved and the *UpdateOnTime()* function is called every two seconds. The saved data have to be accessible to the external world hence *CAN_Tx()* is called, the time spent in ready to drive state in send via CAN only if a particular signal is received. This technique is a good tradeoff between functionalities and bus utilization. For debug purpose in order to understand the current state of the finite state machine *Debug_CAN_Tx()* is called and send a message containing the current state of the finite state machine, the status of cockpit LEDs, and the value of the generated PWM for drive fans. In fact the DashBoard is also able to drive three PWM channels to supply battery pack fans, radiator fans and cooling pumps. This is handled directly via interrupt receiving via CAN the value in percentage of the PWM and applying instantly the changes.

4.6 TLBoard - Safety Module

The TLBoard's firmware is due to debug hence the microcontroller reads the status of the hardwired electronics and writes on CAN. The unit is also capable of reading analog sensors as the SensorBoard do. This permits acquisition on the rear side of the vehicle. The *Core()* function is reported below.

```
/*Core TLBoard*/
void CoreTLBoard(void)
{

        LedBlinking(LED_G_GPIO_Port, LED_G_Pin , 1000);
        //Blink green led

        SaveADCValues(2);
        /*Read from DMA and save 10 samples x channel*/

        FilterADCValues(50);
```

[3] A power cycle means switch off the low voltage system and switching back on.

```
12          /*Mean value for 10 samples each 2ms */

14          BrakeLightRoutine(100);
            /*Check BrakeLight status*/
16
            CAN_Tx(20);
18          /*Send data via CAN*/

20
}
```

The first set of functions are exactly the same as the SensorBoard, the acquisition implementation is the same. The first unknown function is *BrakeLightRoutine()*. Since positioned on the rear side of the vehicle is capable of driving the brake indicator light that is mandatory due to rules, reading on CAN bus the signal coming from SensorBoard and corresponding to the brake pedal position. This sensor is the most precise and the light must be turned on even if the brake is under slightly pressure.

```
void BrakeLightRoutine(uint32_t delay_100us)
2 {
        static uint32_t delay_100us_last = 0;
4
        if(delay_fun(&delay_100us_last,delay_100us))
6       {
                if((BREAK_PEDAL > BREAKLIGHT_THRESHOLD) || DSPACE_BRAKE)
8               {
                        HAL_GPIO_WritePin(BRAKELIGHT_GPIO_Port, BRAKELIGHT_Pin,
                              ON);
10              }
                else
12              {
                        HAL_GPIO_WritePin(BRAKELIGHT_GPIO_Port, BRAKELIGHT_Pin,
                              OFF);
14              }
        }
16
}
```

The functions reads the value of the sensor and if above a certain threshold switch on the light, otherwise off. The condition that switch the light on is in logic *OR* with a second variable, *DSPACE_BRAKE*. In fact the vehicle is able to perform regenerative breaking and rules imposes to switch on brakelight in both regenerative breaking and normal breaking. The variable is sent via CAN from the VMU. The last function sends debug data and is reported below.

```
/*Send data to CAN BUS*/
2 void CAN_Tx(uint32_t delay_100us)
  {
4       static uint32_t delay_100us_last = 0;

6       if(delay_fun(&delay_100us_last,delay_100us))
        {
8               ...
```

```
10                TxHeader.StdId = 0x20;
                  TxHeader.ExtId = 0x020;
12                TxHeader.RTR = CAN_RTR_DATA;
                  TxHeader.IDE = CAN_ID_STD;
14                TxHeader.DLC = 1;
                  TxHeader.TransmitGlobalTime = DISABLE;
16
                  BSPD_ERR = HAL_GPIO_ReadPin(BSPD_ERR_GPIO_Port, BSPD_ERR_Pin);
18                IMD_ERR = HAL_GPIO_ReadPin(IMD_ERR_GPIO_Port, IMD_ERR_Pin);
                  BMS_ERR = HAL_GPIO_ReadPin(BMS_ERR_GPIO_Port, BMS_ERR_Pin);
20                COCKPIT_GREEN_LED = !HAL_GPIO_ReadPin(COCKPIT_GREEN_LED_GPIO_Port,
                        COCKPIT_GREEN_LED_Pin);

22                TxData[0]= (BSPD_ERR<<3) | (IMD_ERR<<2) | (BMS_ERR<<1) |
                        (COCKPIT_GREEN_LED);

24                CAN_Msg_Send(&hcan, &TxHeader, TxData, &TxMailbox, 30);
                  /*Message 3 with Error Byte*/
26
        }
```

The message reported above is the *ERROR_BYTE* mentioned in 4.5. The byte is composed by four errors shifted by one position; with this format is easy to perform bitwise masks. This function also sends acquired signals from analog channels in messages with others *ID*.

4.7 BMS Low Voltage Module

The software module for the BMS for the low voltage system is able to perform different tasks.

```
  /*Thresholds*/
2 extern uint16_t     UV_threshold;
  extern uint16_t     OV_threshold;
4 extern uint16_t     TEMP_threshold;
  extern uint32_t     cell_voltage[8];
6 extern uint16_t     cell_temp_raw[8];
  extern uint16_t     cell_temp[6];
8 extern bool         cell_voltage_fault[8];
  extern bool         cell_temp_fault[8];
10 extern uint16_t        high_cell_voltage;
  extern uint16_t         low_cell_voltage;
12 extern uint16_t     batt_voltage;

14 /*Error status*/
  extern bool faults;
16 extern bool faults_msk;
  extern bool charging;
18 extern bool balancing;
```

Above are reported the main variables used to perform its tasks: under voltage and over voltage threshold, temperature threshold, the voltages values of each cell, the temperature of each cell, the highest voltage and the lowest, the total battery

voltage and some parameters used to set in charging mode, in balancing mode and in fault mask mode. Faults array are used (cell_voltage_faults and cell_temp_fault) in order to determine which cell is faulty and the array is filled as follows.

```
        /*Check voltage thresholds*/
        for(int i = 0; i < 7; i++)
        {
                /*Check OV or UV error*/
                if(cell_voltage[i] > OV_threshold || cell_voltage[i] <
                      UV_threshold )
                {
                   cell_voltage_fault[i] = true;
                }
                else
                {
                   cell_voltage_fault[i] = false;
                }
        }
```

If the cell voltage is below the undervoltage threshold or above the overvoltage threshold the variable is set *true*. With this method the index of the array correspond to the faulty cell id. The same technique is adopted for cell temperatures and if one of them is true a global variable is set as follow.

```
faults |= cell_voltage_fault[i] || cell_temp_fault[i] || (CURRENT > OVERCURRENT);
```

The current of the battery is measured and the value is saved into CURRENT. An overcurret threshold is used and the previous task is performed continuously in order to have a system with a real time response. The *Core()* function of the BMS is reported below.

```
*Core SensorBoard*/
void CoreBmsLV(void)
{

        LedBlinking(G_LED_GPIO_Port, G_LED_Pin , 100);

        if(faults)
        {
                LedBlinking(R_LED_GPIO_Port, R_LED_Pin , 100);
        }
        /*If fault, led should be orange*/
        HAL_Delay(200);

        /*Poll 8 sens + 8 aux*/
        if(Uart_Send_CMD(0x81, 0x00, 0x02, 0x01, 0xe8, 0x9c, 0x00, 0x00, 0x00,
              0x00, 6,  10) != HAL_OK)
        {
                Error_Handler();
        }

        /*Read 8 sens + 8 aux*/
        if(Uart_Receive_CMD(uart_rx_cmd, 35,  100) != HAL_OK)
        {
```

```
                Error_Handler();
24          }
            /*END OF CO-PROCESSOR COMMANDS*/
26
            batt_voltage = 0;
28          /*Fill voltage array with converted values, compute total, lower and
                  higher voltage*/
            for(int i = 0; i < 7; i++)
30          {
                cell_voltage[i] = (uart_rx_cmd[2*i+3] << 8);
32              cell_voltage[i] = (cell_voltage[i] | uart_rx_cmd[2*i+4]);
                cell_voltage[i] = (cell_voltage[i]  * 5000 ) / 65535;
34              batt_voltage += cell_voltage[i];

36              low_cell_voltage = 5000;
                if(low_cell_voltage >= cell_voltage[i])
38              {

40                      low_cell_voltage = cell_voltage[i];

42              }

44              high_cell_voltage = 0;
                if(high_cell_voltage <= cell_voltage[i])
46              {

48                      high_cell_voltage = cell_voltage[i];

50              }

52          }

54          /*Fill temp array with converted values*/
            for(int i = 0; i < 8; i++)
56          {
                cell_temp_raw[i] = (uart_rx_cmd[2*i+17] << 8);
58              cell_temp_raw[i] = (cell_temp_raw[i] | uart_rx_cmd[2*i+18]);
                cell_temp_raw[i] = 100.0 * (1.0 / (1.0/298.0 +
                      (1.0/BETA)*log(1.0/(65535.0/(float)cell_temp_raw[i] - 1.0))
                      ) - 273.0);
60          }

62          /*END CONVERSION VOLTAGEs and TEMPERATURE*/
            ...
64
            if(CURRENT < CURRENT_OFFSET_CHARGING)
66          {
                charging = true;
68              HAL_GPIO_WritePin(R_LED_GPIO_Port, R_LED_Pin, GPIO_PIN_SET)
                /*Turn on yellow led*/
70
            }
72          else
            {
74              charging = false;
                HAL_GPIO_WritePin(R_LED_GPIO_Port, R_LED_Pin, GPIO_PIN_RESET)
76              /*Turn off yellow led*/

78          }

80
            /*Check current and temperature threshold*/
```

```
82          for(int i = 0; i < 7; i++)
            {
84                  /*Check OV or UV threshold*/
                    ...
86          }
            for(int i = 0; i <6; i++)
88          {
                    /*Check over temperature*/
90                  ...
            }
92
            CAN_Tx(2500);
94          /*Send via can*/

96          /*Open or Close vehicle relays*/
            if(((faults == false) && (charging == false)) || (faults_msk == true))
98          {
                    /*CLOSE RELAIS*/
100
                    HAL_GPIO_WritePin(RELAY_CMD_GPIO_Port, RELAY_CMD_Pin,
                          GPIO_PIN_SET);
102
            }
104         else
            {
106                 /*OPEN RELAIS*/
                    HAL_GPIO_WritePin(RELAY_CMD_GPIO_Port, RELAY_CMD_Pin,
                          GPIO_PIN_RESET);
108         }

110 }
```

The BMS led positioned on the board behaves as others with new color, in fact if the battery is in charging mode the led becomes orange (switching on red and green). The task performed at beginning are commands needed to be sent to the co-processor via uart in order to request voltages and temperatures. Once received values are converted into voltage since are coming from the internal ADC with 4.1.

$$cell\_voltage = \frac{ReadValue * V_{FR}}{2^{N_b}} = \frac{ReadValue * 5}{2^{16}} \tag{4.1}$$

The number of bits N_b is fourteen but during the transmission via UART two byte are received for a total of sixteen bits. Hence the reason of sixteen used for the conversion. Then the temperature values are acquired first into voltage then into degrees as in equation 4.2

$$cell\_temp = \frac{1}{\frac{1}{298} + \frac{1}{\beta} * log(\frac{1}{\frac{2^{N_b}}{ReadValue} - 1})} - 273 \tag{4.2}$$

After the data conversion, the current read by the sensor is compared with a minimum threshold, if below the current is flowing through the battery and a variable is set as *true* then the red led is switched on changing color to orange and vice-versa. Once performed this check at the end the task decides to open or close

the battery relay with the possibility of fault masking sending a command via CAN. This task is repeated indefinitely every 200 milliseconds.

4.8 PC Host Module

The host pc module is an application that runs on Windows able to send commands and perform debug via CAN. This module that is mandatory to upgrade unit's firmware is written in C++ using Qt and is composed by three treads. The main thread is the graphical interface, the second performs CAN broker between the application and the low level driver and the last is a dedicated thread that uses OpenBLT libraries to upgrade firmwares. The graphical interface is reported below in figure 4.6.

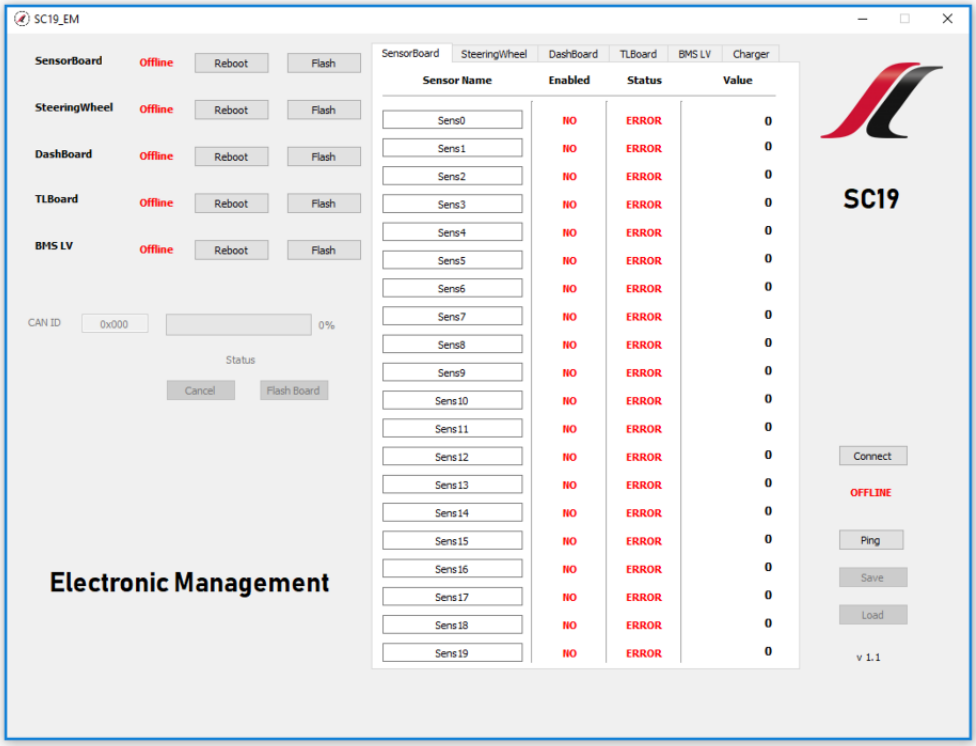

Figure 4.6: Host PC

The tool connects via CAN and reads all data to update the graphical interface. Is possible to upgrade individually unit's firmware and a dedicated sheet is available for each unit. For example as in figure 4.7 the DashBoard internal variable are

shown as finite state machine status and error byte received. Is is also possible to set a certain value for the PWM and send instantly to the unit.

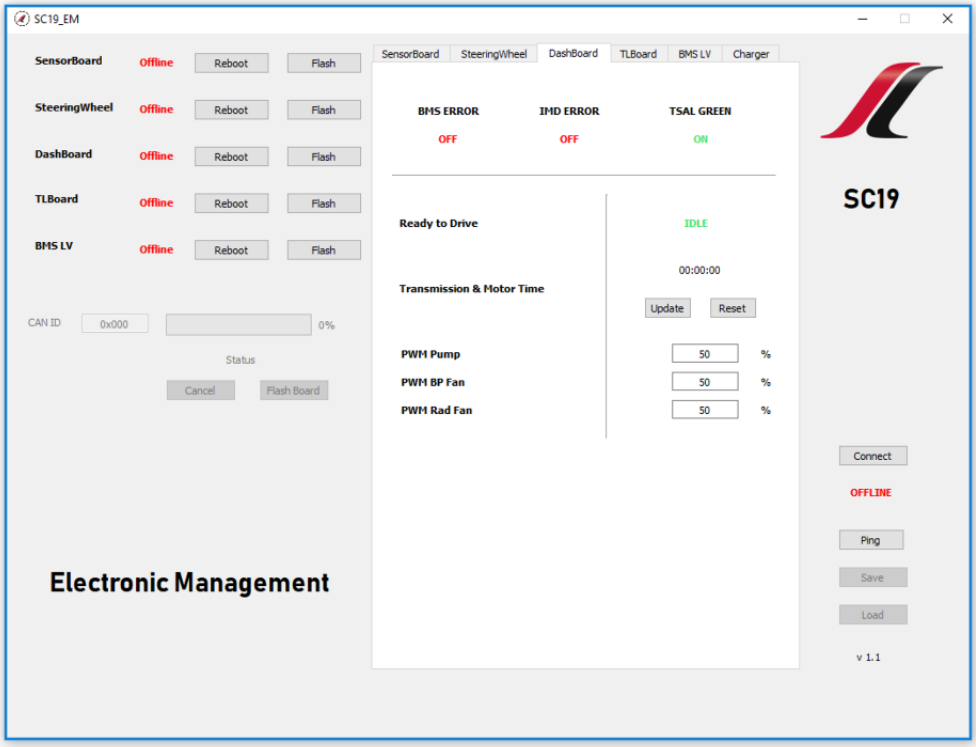

Figure 4.7: DashBoard tile

This tool is also useful when the high voltage battery has to be charged: in fact the battery have the be unmounted from the vehicle and positioned in a custom battery charger. The high voltage BMS in fact needs some commands to close AIRs hence a the tool is able to perform this task. As in figure 4.8 the tool have some buttons in order to start the charging procedure and show the state of charge in real time, the temperature of the highest cell, the value of the charging current and the total battery voltage.

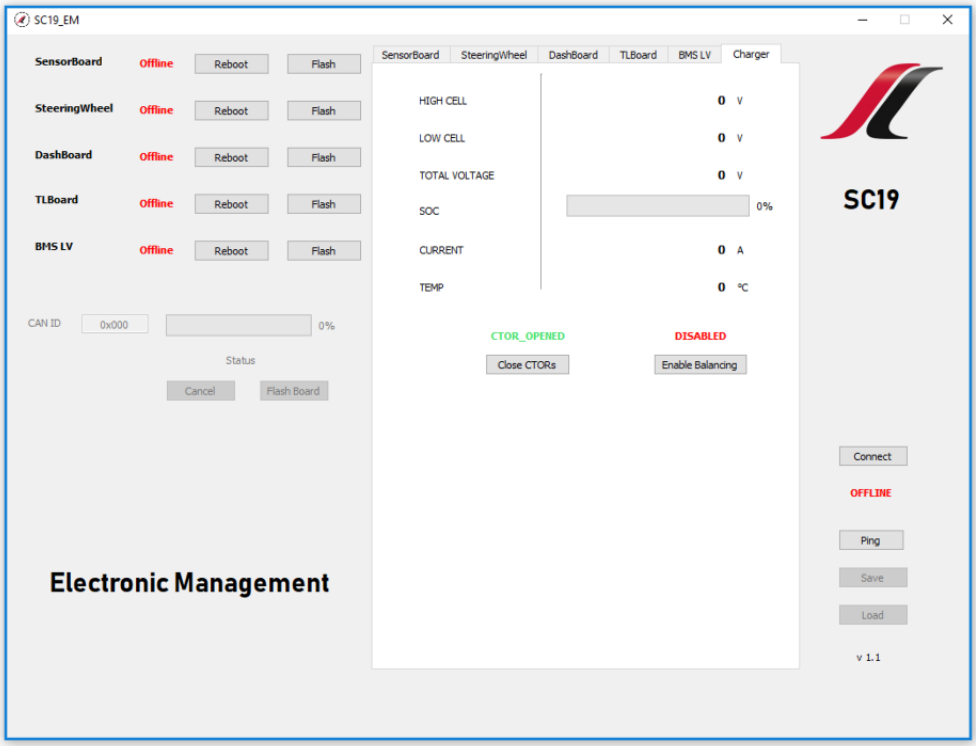

Figure 4.8: Charger tile

One of the most useful feature this module implements is the ping mode: the tool is able to perform a ping to each unit and waits for a response. This result useful in the beginning debug phase in order to identify if the unit is online or not.

Appendix A

Control Area Network BUS

The Controller Area Network (CAN) as described in [2] is a multi-master communication protocol initially developed by BOSH. Originally created for automotive industry for replacing complex communication systems with a only two wire bus. The messages are broadcasted to the entire network providing data consistency in every node of the system. The communication protocol defines each node must wait a specific amount of time before attempting to send a message and collisions are resolved trough arbitration based on priority of each message in the identifier filed: the highest priority identifier wins bus access. Standard identifiers are 11-bit depth providing 2^{11} different identifiers instead of extended identifier that are 29-bit depth with at least 2^{29} different identifiers. Messages can be classified into two categories: *Data Frame* and *Remote Frame*. The former are messages containing data byte, the latter are messages without data used to solicit the transmission of a corresponding data frame. Data frame with standard identifiers are formatted as in figure A.1.

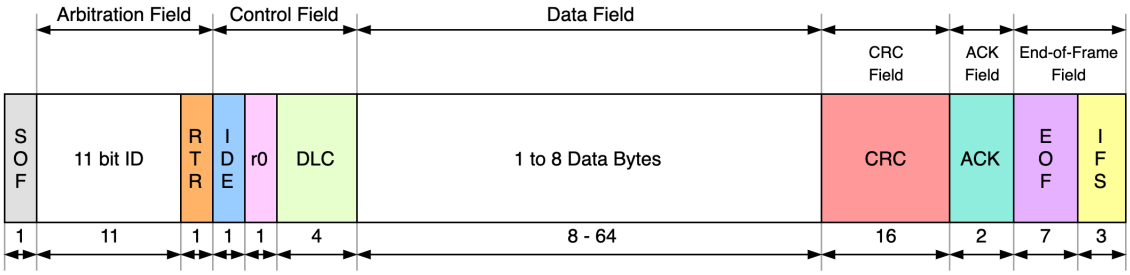

Figure A.1: Data Frame with 11-bit identifier

Remote frame are formatted as figure A.2.

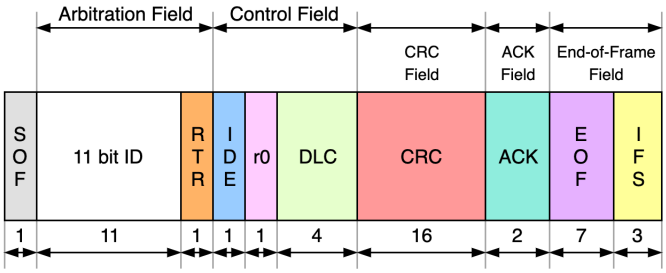

Figure A.2: Remote Frame with 11-bit identifier

As shown above bit field are the same for both data and remote frame: fields will be explained in the following.

- **SOF**: (Start of Frame) marks the beginning[1] of data and remote frames with a dominant[2] bit.
- **Arbitration Field**: includes message ID and RTR bit. Identifiers are 11 or 29 bit and the first bit is the MSB. In case of an extended message the identifier is split into two pieces: as in figure A.3 block **A** contains the first 11 bits of the ID (from 28 to 18) and the second block **B** contains the remaining (17 to 0). If RTR = 0 the message is a data frame otherwise a remote frame. For extended ID the IDE is present in Arbitration Field, if equals to one specify an extended identifier, otherwise standard[3].
- **Control Field**: contains the DLC and a reserved bit *r0*. In case of a standard ID also the IDE field, but for extended a second reserved bit *r1* is present.

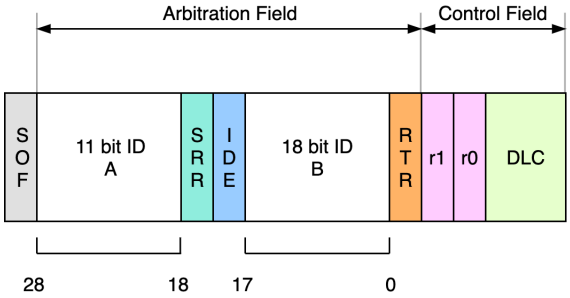

Figure A.3: Extended Frame

[1]The falling edge of SOF identifies the attention of a node to access the bus and serves as a synchronization mechanism between nodes

[2]Zero logic since is like open collector principle

[3]The position of IDE bit must be considered, since in both standard or extended is the same

- **Data Field**: contains from 1 to 8 bytes of data.
- **CRC Field**: this field contains the CRC frame 15 bits deep and one bit of delimiter that is always recessive.
- **Acknowledgement Field**: contains one bit of acknowledge and one bit of delimiter that is always recessive. This field serves as confirmation of a successful CRC check by the receiving nodes in the network.
- **End-of-Frame Field**: each frame is terminated by a sequence of seven recessive bits (EOF) plus an interframe space of three bits (IFS). During interframe bits nodes are not allowed to transmit on the bus.

Physical signaling layers generally are implemented into any controller. Connection to the physical layer is implemented through a transceiver as shown in figure A.4.

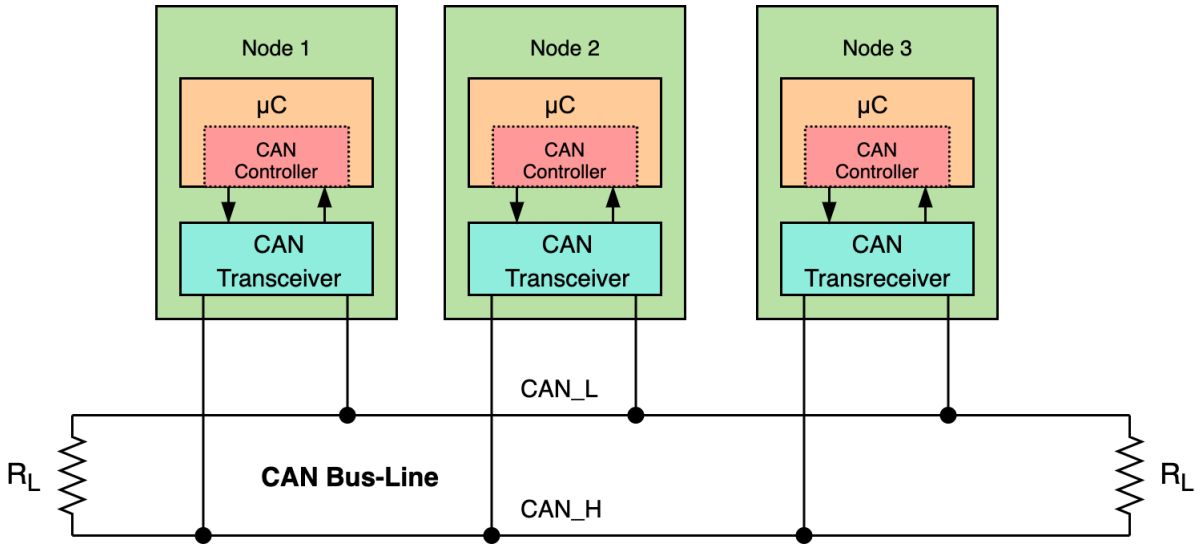

Figure A.4: Physical connection of a CAN Network

Signaling are differential which provides noise immunity and fault tolerance. The signal is balanced, meaning that the current flowing in each signal line is equal but opposite. This solution reduce noise coupling and allows for high rates over twisted pairs. The use of balanced differential together with twisted pair cabling enhances the common-mode rejection and provides high noise immunity. The cable could be both shielded or unshielded with 120Ω of characteristic impedance; the line have to be terminated at both ends with a 120Ω resistors in order to mach the impedance to avoid signal reflection as shown in figure A.4. Two signal lines are labeled *CAN*

H and *CAN L*, in a recessive state are biased to 2.5v in order to have a typical 2v different between.

Glossary

Acceleration The vehicle's acceleration from a standing start is measured over a 75 metre straight 4

ADC Analog-to-Digital Converter 19

AIR Accumulator Isolation Relay 27, 28, 29, 30, 32, 46, 53

AMS Accumulator Management System 28

Autocross The car traverse a kilometre-long with straights, curves, and chicanes. A fast lap time is a sign of high driving dynamics, precise handling and good acceleration and braking ability 4

BMS Battery Management System 6, 7, 8, 18, 19, 22, 23, 24, 25, 26, 31, 32, 44, 48, 49, 51, 53

BOT Brake over Travel 28

BSPD Brake System Plausibility Device 6, 7, 28

Business Plan Simulation of the presentation of their project by each team in front of potential sponsors 4

CAN Controller Area Network 7, 33, 35, 36, 38, 40, 43, 44, 46, 47, 52, 55

Cost and Manufacturing Analysis of the cost report drafted by each team where are included quantities of materials and components used 4

CRC Cyclical Recovery Checking 38, 57

DC Direct Current 25

DC-to-DC Direct Current to Direct Current Converter 5, 6

DLC Data Length Code 56

Endurance Endurance is the main discipline; over a distance of 22 kilometers the cars have to prove their durability under long-term conditions 4

Engineering Design Car project presentation 4

EV Electrical Vehicle 24

HVD High Voltage Disconnect 28

IMD Insulation Measurement Device 6, 28, 31, 32, 33, 44

LED Light Emitting Diode 8, 43, 44, 45, 46

LV Low Voltage 5, 6, 28

LVMS Low Voltage Master Switch 28

LVS Low Voltage System 5

MSB Most Significant Bit 56

PCB Printed Circuit Board 3, 8, 9

RTR Remote Transission Request 56

Skid Pad The cars must drive a figure of 8 circuit lined with track cones, performing two laps of each circle 4

TS Tractive System 5, 6

TSAL Tractive System Active Light 6, 7, 31, 32, 33

TSMS Tractive System Master Switch 28

UART Universal Asynchronous Receiver-Transmitter 19, 20, 22, 43, 51

VMU Vehicle Management Unit 6, 7, 35, 36, 41, 45, 46, 47

Bibliography

[1] Formula Student *Germany* Rules, 2019 v1.1, Rev-713. Link.

[2] Introduction to the *Controller Area Network*, 2002, Texas Instruments Link.

[3] AUTOMOTIVE CURRENT TRANSDUCER OPEN LOOP TECHNOLOGY DHAB S-145Link.

[4] OrionBMS JuniorLink.

[5] STM32F303xB / STM32F303xC - ARM based Cortex-M4 32b MCU+FPU, up to 256KB Flash+ 48KB SRAM, Datasheet Link.

[6] Common Mode Chokes in CAN Networks: Source of Unexpected Transients, Application Note Link.

[7] Texas Instruments BQ76PL455A-Q1 16-Cell Battery Monito, Datesheet Link.

[8] CAS/CASR/CKSR series Current Transducers, LEM, Datesheet Link.

[9] OpenBLT GNU GPL Bootloader, Link.

[10] STM32CubeMX, initialization code generator Link.
https://www.st.com/en/development-tools/stm32cubemx.html