UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN


MASTER OF SCIENCE IN TELECOMMUNICATION ENGINEERING

MASTER THESIS

# DESIGN AND IMPLEMENTATION OF A HIGH-PERFORMANCE HARDWARE PLATFORM FOR DRIVING MOTOR CONTROL SYSTEMS.

ALEJANDRO GÓMEZ MOLINA

2022

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN


MASTER OF SCIENCE IN TELECOMMUNICATION ENGINEERING

MASTER THESIS

# DESIGN AND IMPLEMENTATION OF A HIGH-PERFORMANCE HARDWARE PLATFORM FOR DRIVING MOTOR CONTROL SYSTEMS.

Author
ALEJANDRO GÓMEZ MOLINA

Tutor
ÁLVARO GUTIÉRREZ MARTÍN

2022

## Abstract

A control system can be defined as a group of resources that can generate response control signals from an external physical stimulus. These signals are capable of controlling and regulating the behavior of a physical system.

Modern control systems make intensive use of hardware and software resources to meet the stringent requirements of these kinds of system. These requirements range from efficient use of software resources to thermal management. As it seems, it would be an exhaustive task, especially for non-expert developers.

Therefore, in view of all these elements, the aim of this final master thesis will be to design and implement a hardware platform and a front-end framework for a complete generic motor control system. The main objective is to create a system capable of handling high-power motor devices and different types of input transductors and a user-friendly configuration front-end.



**Keywords:** PDI, controller, PCB, embedded systems, ROS, brushed motor

## Acknowledgement

I am really grateful to my supervisor Álvaro Gutiérrez Martín for giving me the opportunity to work on such an interesting topic, this thesis could not be completed without his support and full confidence in me and the decisions made. Thank you for all technical and personal advice during the last few years, you were an important part of my path to becoming an engineer.

I also would like to express my special thanks for the gratitude of all RBZ embedded logics team for providing economical, technical and physical support. I am delighted to have had the opportunity to work with an impressive group of experts such as Jesús, Dani, Diego, Gonzalo, Sergio, and all other members of the RBZ family. Thank you for sharing your knowledge and experience with me and making me feel like a member of the team.

Last but not least, I would like to acknowledge the unconditional support, confidence, and love given by my family and friends. Because their love and guidance gave me the strength for compelling all my life projects.

# Contents

# List of Figures

# List of Tables

# Lists of acronyms

| | |
|---|---|
| **AC** | Alternating Current |
| **ADC** | Analog to Digital Converter |
| **API** | Application Programming Interface |
| **BJT** | Bipolar Junction Transistor |
| **BSD** | Berkeley Software Distribution |
| **BSP** | Board Support Package |
| **CAN** | Controller Area Network |
| **CCW** | Counterclockwise |
| **CPR** | Counts Per Revolution |
| **CPU** | Central Processing Unit |
| **CRC** | Cyclic Redundancy Check |
| **CSMA/CD** | Carrier Sense Multiple Access with Collision Detection |
| **CW** | Clockwise |
| **DC** | Direct Current |
| **DDS** | Data Distribution Service |
| **eFSM** | Extended Finite State Machine |
| **ESD** | Electrostatic discharge |
| **FPGA** | Field Programmable Gate Arrays |
| **FSM** | Finite State Machine |
| **FSR** | Full Scale Range |
| **GPIO** | General Purpose Input/Output |
| **HS** | High Side |
| **HW** | Hardware |
| **I2C** | Inter-Integrated Circuit |
| **IC** | Integrated Circuit |
| **ID** | Identificator |
| **IEEE** | Institute of Electrical and Electronics Engineers |
| **IoT** | Internet of Things |
| **IP** | Internet Protocol |
| **IRQ** | Interrupt Request |
| **LAN** | Local Area Network |
| **LCD** | Liquid-Crystal Display |
| **LDO** | Low dropout |
| **LED** | Light-EmittingDdiode |
| **LS** | Low Side |
| **LSB** | Less Significant Bit |
| **LTL** | Linear Temporal Logic |
| **LwIP** | Lightweight IP |
| **MAC** | Medium Access Controller |

| | |
|---|---|
| **MAN** | Metropolitan Area Network |
| **MCU** | Microcontroller Unit |
| **MII** | Media-Independent Interface |
| **MIT** | Massachusetts Institute of Technology |
| **MOSFET** | Metal-Oxide-Semiconductor Field-Effect Transistor |
| **MSB** | Most Significant Bit |
| **NRZ** | Non-Return Zero |
| **OSI** | Open Systems Interconnection |
| **PCB** | Printed Circuit Board |
| **PID** | Proportional–Integral–Derivative |
| **PPR** | Pulses Per Revolution |
| **PSRR** | Power Supply Rejection Ratio |
| **PWM** | Pulse Width Modulation |
| **RAM** | Random Access Memory |
| **RMII** | Reduced Media-Independent Interface |
| **ROM** | Read-Only Memory |
| **ROS** | Robot Operating System |
| **RPM** | Revolutions Per Monite |
| **RT** | Real-Time |
| **RX** | Reception |
| **SD** | Secure Digital |
| **SDK** | Software Development Kit |
| **SMPS** | Switching mode power supplies |
| **SOF** | Start of a frame |
| **SoM** | System on a Moduel |
| **SPI** | Serial Peripheral Interface |
| **SUA** | Sudden Unexpected Acceleration |
| **SW** | Software |
| **TCP** | Transmission Control Protocol |
| **TPL** | Third Party Library |
| **TX** | Transmission |
| **UART** | Universal Asynchronous Receiver / Transmitter |
| **UDP** | User Datagram Protocol |
| **USB** | Universal Serial Bus |
| **WSL** | Windows Subsystem for Linux |
| **XRCE-DDS** | eXtremely Resource Constrained Environment Data Distribution System |

# Chapter 1

# Introduction and Goals

## 1.1 Introduction

Control systems are part of our daily life. Nearly all electronics which interact with a physical environment have almost one sort of control system to regulate their behavior. Any environment is constantly changing, the change is constant in our world, but the problem is that there is not a way to predict where, when, or how it will occur. It is evident that it is impossible to create systems programmed to handle all possible changes, there are infinities. However, it is possible to create resilient systems with the ability to automatically adapt its behavior in concordance with the external perturbance.

The importance of these devices becomes clear when realize that they are in charge of the safety of millions of human lives: airplane controllers, brake systems, or even surgical appliances are controlled by these systems. Consequently, control systems must meet stringent requirements.

Creating a control system is not an easy task, every system must be analyzed and modeled to create a proper controller. Furthermore, these modeled systems must be implemented in real hardware and software platforms where non-idealities are always present. These platforms also must meet strict requirements in the range from efficient use of resources to thermal management. As it seems, it would be an exhaustive task even for senior control developers.

This thesis has been developed with the key idea of simplifying the development process of brushed motor control systems. It creates a generic control hardware platform and an abstraction layer to interact with the hardware in an easy way.

In this first chapter some concepts related to brushed motor control systems will be introduced briefly. Next, it will be introduced the grounds of and the motivations of this thesis and, finally, a quick summary of the whole document structure will be done.

## 1.2 Control systems

A control system is defined as a set of devices that manages, commands, directs, or regulates the behavior of other devices or systems to achieve a desired result. The main feature of a control system is that there should be a cause-effect relation for the components of the system.There should be a mathematical relation between the input and the output of the system [10]. This relation between the input and the output is known as transfer function or network function.

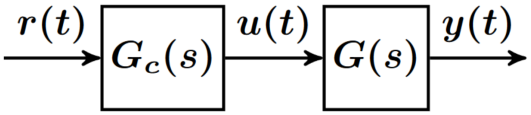



Figure 1.1: Diagram of open-loop control system [38].

A basic control system can be represented by the block diagram of Figure 1.1. It is composed by two main blocks: a plant (or system) to be controlled with a transfer function $G(s)$ and a controller with a transfer function $G_c(s)$, whose function is to generate the necessary control signal $u(t)$ for the plant. This system is called open-loop control system due to the control signal $u(t)$ depends only on the stimulus signal $r(t)$, not being affected by the output signal $y(t)$ or, what it is the same, the control action is independent of the desired output.

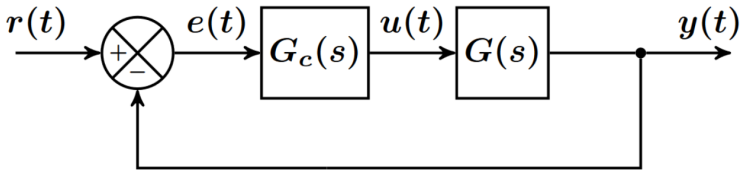



Figure 1.2: Diagram of close-loop control system [38].

A more complex system can be created by adding a feedback branch from the output to the input as shown in Figure 1.2. In this type of systems, the input of the controller is the difference between the stimulus signal $r(t)$ and the feedback signal $y(t)$, also called error signal $e(t)$. These systems are called close-loop control systems.

Control systems are designed with different purposes and requirements. However, all of them share a common essential requirement: they must be stable. Furthermore, there are some usual goals when designing a control system [38]:

- **Steady state specifications:** The error signal $e(t)$ must be zero when t

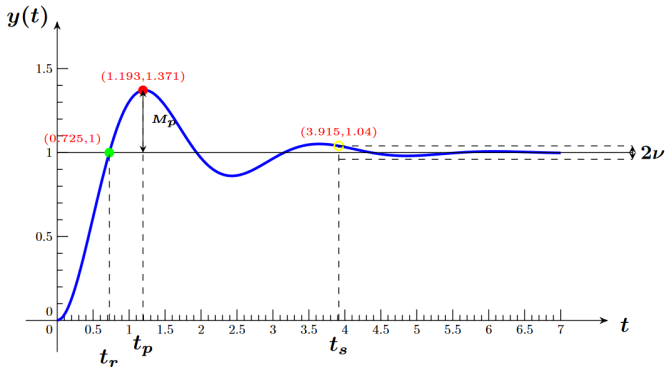



Figure 1.3: Response to a unity step function in a second-order system [38].

approaches infinity $t \to \infty$, where $e(t) = r(t) - y(t)$ and $r(t)$ is the reference signal desired in the output. Ideally, when the system meets this requirement, it is said that the system is in steady state. However, for practical reasons, the system is considered in a steady state when, for a time $t \geq t_s$, the value of $|y(t)| \leq v$, where $v$ is a tolerance commonly expressed as percentage. In other words, the system is in steady state when the output is within a defined error band.

The time elapsed from the application of an input stimulus to the time in which the output remains within the error band is known as the settling time $t_s$. The settling time is just another parameter defined in the transient state.

- **Transient state specifications:** As shown in Figure 1.3, some of the main specifications to take into account for second-order systems are:
  - **Settling time ($t_s$):** necessary time to reach the steady state.
  - **Overshoot ($M_p$):** maximum value at the output.
  - **Pea time ($t_p$):** time elapsed to reach the maximum output value.
  - **Rise time ($t_r$):** time taken by the output to cross the reference value for the first time.
- **Perturbation suppression:** External perturbations should not affect the system in steady state or, at least, when $t \to \infty$.

### 1.2.1 Control: digital control systems

Digital control systems are similar to analog control systems, but here the analog control is replaced by a digital processor. These systems are also known as hybrid time systems since they use continuous and discrete parts at the same time. The superior performance, low cost, and flexibility of the design explains the growing popularity of these systems over analog controllers [28]. Figure 1.4 shows the typical structure of a digital control system.

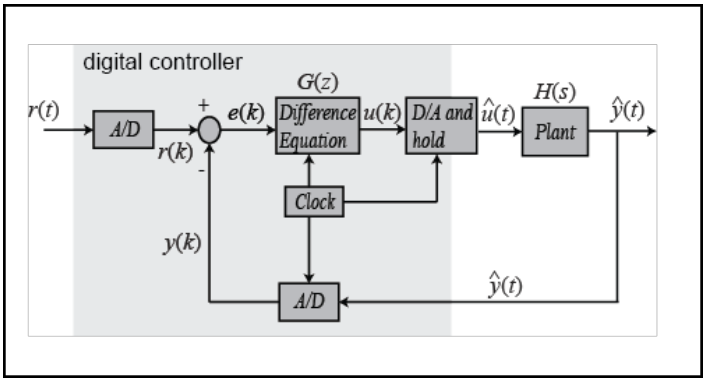



Figure 1.4: Typical diagram of a digital control system [30].

Following its discrete nature, digital control systems use analog-to-digital converters (ADC) to discretize the input signals and digital-to-analog converters to generate the necessary continuous signals to control the plant. Normally, signals are sampled with a sampling period $T \in \mathbb{R}^+$.

## 1.3 Controller plant

The signal generated by the controller is applied to the plant, which is the real actuator to be controlled. The physical nature of this plant defines how the actuator reacts to a given control signal. The design of the controller relies on the plant behavioral model, and this model is made creating a mathematical description of its internal structure. This section describes the internal structure of brushed direct current motors, the plant on which this project is focused.

### 1.3.1 Direct current motors

Direct current motors (DC motors) are electromechanical devices designed to transform electrical energy into movement. This type of motor is widely used in electronic applications such as automotive control systems, robotics, and home appliances where speed or position control is needed due to its cost efficiency and simplicity.

A brushed DC motor is a type of DC motor that is mainly composed by two parts: the outside casing, known as stator, with two permanent magnets attached around each one with different polarity, and the inner dynamic part known as rotor. The rotor is made up of at least three windings connected to metal plates known as armatures. This kind of motor gets its name from the carbon brushes that provide the energy from the supply to the windings via the commutator. The commutator is made up of small electrical contacts or segments connected to different windings [27]. In movement, the brushes enter in contact with different segments, energizing the windings, and therefore generating a dynamic magnetic field inside the motor. Figure 1.5 illustrates the different parts of a motor.

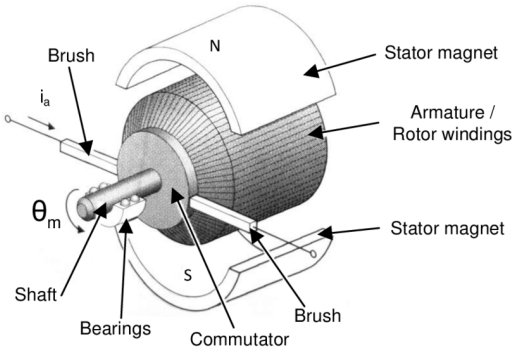



Figure 1.5: Schematic of a brushed direct current motor [42].

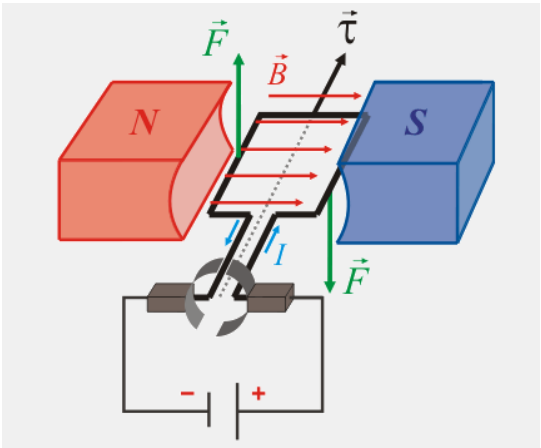



Figure 1.6: Working diagram of a electric motor [4].

A brushed DC motor generates torque from DC power supply applying the Lorentz principle. This principle says that any current-carrying conductor placed within an external magnetic field experiments a force whose magnitude depends on the relative orientation between the current-carrying conductor and the field. A working diagram of a DC motor is shown in Figure 1.6.

### 1.3.2 PWM Control

The speed of DC motors and the direction of rotation are controlled using PWM signals. PWM stands for Pulse Width Modulation, and it is a modulation technique for encoding the amplitude of a signal into the duty cycle of another periodic signal used as a carrier (a square signal, for instance). This modulation technique is typically used in communication systems or to control the amount of energy supplied to a load. The duty cycle is defined as the relation between the time in which the signal is active and the period of the signal [62].

Due to mechanical inertia and coil inductance that avoid instant changes, DC motors only react to DC signals. When a high-frequency signal is applied, the motor reacts only to its DC component, also known as the average voltage. The average

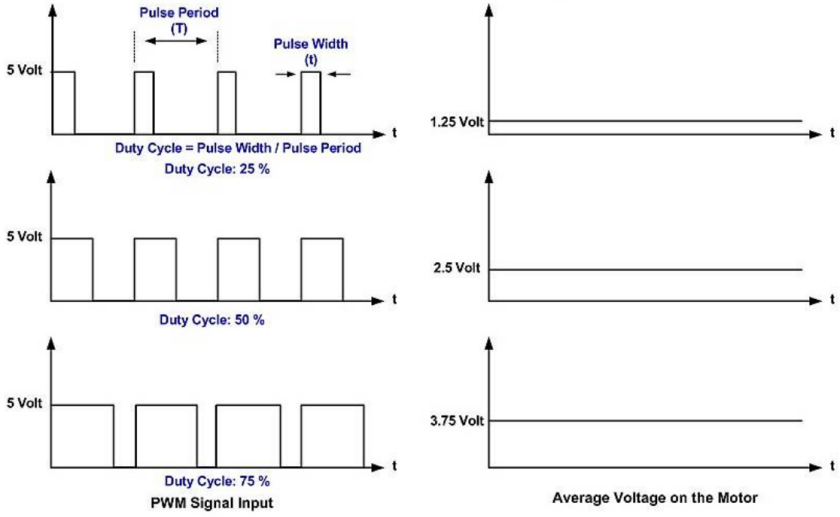



Figure 1.7: PWM timing diagram [62].

voltage of a PWM signal is given by its duty cycle following Expression 1.1. Hence, by modulating the width of the duty cycle of the PWM signal, the average voltage supplied to the motor is modified, and therefore its speed. Figure 1.7 shows some examples of the relation between duty cycle and average voltage.

$$D = \frac{\tau}{T} \times 100\% \tag{1.1}$$

$$V_{avg} = V_{max} \times D \tag{1.2}$$

Where $D$ is the duty cycle, $\tau$ is the duration of the positive state, $T$ is the period of the signal, $V_{max}$ is the peak voltage of the signal and $V_{avg}$ is the average voltage.

PWM provides many advantages to the control system; for example, it can be easily controlled and generated by a digital system. Furthermore, one of the most important advantages is that it always provides the maximum torque to the motor independently of the average voltage. A DC motor torque is determined by the amount of current supplied and depends on the voltage applied to the motor (Ohm's law). Given that a PWM signal always provides the maximum voltage, the torque is always maximum.

### 1.3.3 Rotatory Encoders

Incremental position sensors, commonly known as incremental encoders, are devices used to measure the speed and angular position of rotating objects by detecting discrete steps of linear or angular displacement [35]. Due to its low cost and ability to provide easily interpreted output signals, these devices are widely used in the industry to measure revolutions per minute (RPM), direction, and position of motors. Encoders are made using three main technologies: optical, magnetic, or capacitive.

The simplest form of incremental encoder has a single output line on which a signal toggles to indicate each position increment [35]. Given that this single signal does not provide information about the direction, a second line is used to provide that information, giving, as a result, a dual-channel device. These two outputs are also called quadrature outputs since they are out of phase at 90 degrees [39]. As shown in Figure 1.8, the position, clockwise (CW) or counterclockwise (CCW), depends on which channel phase leads.

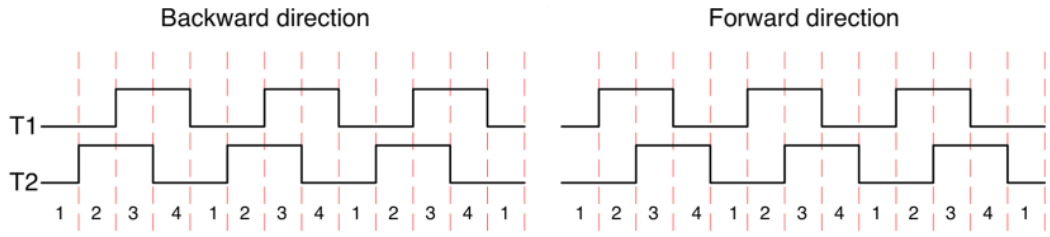



Figure 1.8: The square waves emitted by a quadrature encoder [39].

The angular resolution of an encoder is defined by its CPR (counts per revolution) or by its PPR (pulses per revolution). The CPR is the number of changes on both channels in one revolution; in contrast, the PPR is the number of pulses in a single channel. In a quadrature encoder, CPR is achieved by multiplying PPR by four, as shown in Figure 1.9.

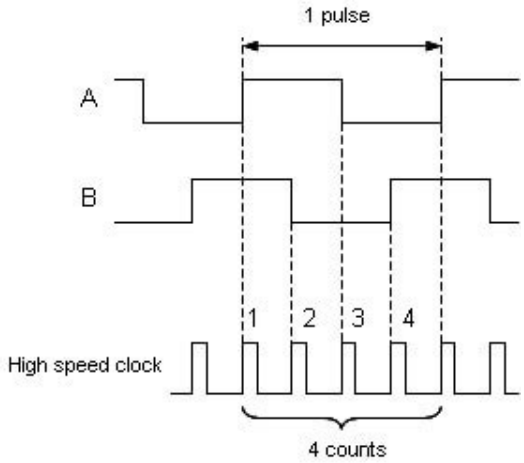



Figure 1.9: Difference between CPR and PPR [47].

Some microcontrollers have special hardware to drive incremental encoders that allows efficient use of encoders with high CPR. This feature will be exploited in this project.

## 1.4 Grounds for and objectives

In view of the complexity of a digital control system and the large number of technical variables to take into consideration when designing a physical control system, the main goal of this thesis is to design and implement a hardware control platform for a generic motor control system. This platform is created with the aim of simplifying and speed-up the modeling and design process by the creation of an abstraction layer between the user and the hardware.

This platform will be capable of driving high-power motor devices and different types of input transductors, both digital and analog. The system will be designed to be scalable, reusable, and powerful and is aimed at control engineers in laboratory applications.

The design process will follow a series of necessary steps to achieve the proposed goals. These steps will not be followed strictly in a linear way, but will be followed using an iterative scheme allowing to change, improve, and include new characteristics and requirements to the project. The steps are:

- Study of the background of control systems.
- Capture of user requirements.
- Definition of hardware and software requirements.
- Translation of the requirements into implementable design blocks.
- Implementation of the blocks.
- Unit testing.

## 1.5 Document structure

This document is structured in five main blocks. In the first chapter of the present document, the global theoretical background, as well as motivations and goals are introduced.

The second chapter presents the hardware design process. This chapter describes the blocks of the hardware platform, how they are made, and the relation among them, paying special attention to the component selection. This is followed by an explanation of the circuit board design and fabrication process.

The third chapter describes the software design process from requirement gathering to its implementation on the hardware platform. In this chapter, the decisions made to meet the strictly time requirements needed in control systems are described.

The fourth chapter contains the testing process for all different blocks of hardware and software parts. In this chapter, an incremental testing process is applied to reduce the probability of a critical failure and to narrow the search of existing bugs.

Finally, in the fifth chapter, a critical analysis of the targets achieved is made. Using this analysis and the results of the testing process, improvements and new requirements are proposed for future versions.

# Chapter 2

# Hardware Design

## 2.1 Introduction

According to Oxford [45], hardware can be defined as the machinery and electronic part of a computer system. Indeed, it is the physical support and the main constraint of any computer system, that is to say, the whole system is first limited by the hardware performance.

Unlike software, hardware usually requires a more tedious creation process, which involves a great number of physical resources in each step. Manufacturing is a slow and complex process that uses high-tech machinery in specialized fabrics. Therefore, the time taken from the design of the prototype to the moment in which it is manufactured and fully tested is relatively long. On top of that, it is evident that any mistake done during the design may incur a new iteration of the whole process and, consequently, a break of weeks in the production process. In the vast majority of cases, a hardware problem cannot be fixed with a simple update, it will need a physical intervention.

As a physical device, the hardware is exposed to a number of physical variables (such as temperature, humidity, light, etc.) and must work in potentially hostile environments, mostly crowded by hundreds of other electronic devices. Designers must take this into consideration to create hardware capable of withstanding not only to natural perturbations, but also incoming disturbances from other devices, all of this without disturbing other electronic devices. This is not a simple choice, all these requirements are gathered in directives that consumer electronics must meet and certify.

As if this were not enough, the design process has many other factors that affect it: cost, form factor, size, normative, power consumption, and the manufacturing process are some common factors. In addition, there are external factors, such as global economy, natural disasters or political movements that, although they do not directly concern the hardware, can affect the whole supply chain. An example of this is the Sars-Cov-2 pandemic, and the following chip stock break suffered worldwide. This shortage is expected to continue within the next two years and, as a result, is changing the way in which the hardware is designed.

Overall, this chapter describes the hardware design process from requirements to the manufacturing and testing process. Hardware has been divided into design blocks lumping together parts with similar requirements or functionalities. It will be explained how these blocks are turned into schematics and then into printed circuit boards. Finally, the manufacturing and testing process will be described from an industrial perspective.

## 2.2 Requirement gathering

Requirements gathering is the procedure by which a designer transforms the user needs into a group of potentially implementable features. This process requires constant fluid and bidirectional communication among the parties involved. In other words, it can be seen as the translation of the user's wishes into a list of real technical requirements.

This project has started with the aim of covering some of the most important needs in control system laboratories. After many meetings with the parties involved, the hardware requirements shown in Table 2.1 have been collected.

## 2.3 Common concepts

### 2.3.1 Transmission lines

When the frequency of signals traveling through an electric wire is nearly direct current, it can be assumed that the electric voltage across the conductor is the same in all points. However, if the frequency of signals is high and at the same time the length of the trace is long enough, the trace cannot be treated as ordinary. To determine whether a trace is electrically 'long' or 'short', it is necessary to know the wavelength ($\lambda$) of the signal that can be calculated using Expression 2.1. As a simple rule of thumb, if the trace length is greater than 1/10 of the wavelength of the signal, it must be considered a transmission line [9].

$$\lambda = \frac{C}{f} \tag{2.1}$$

Where $C$ is the speed of light and $F$ is the frequency of the signal.

A transmission line is a physical environment or structure that allows the confined and guided transmission of an electronic wave between two points, typically between a source (origin) and a load (destination). It is made of conductors and dielectrics whose geometry defines the way in which the waves are transmitted.

Table 2.1: Hardware requirements.

| ID | Requirement |
|---|---|
| | **Form factor** |
| **HF-1** | The prototype will be formed by multiple boards forming a stack structure. |
| **HF-1.1** | There must be at least two boards to divide the control block from the high-power block. |
| **HF-1.2** | The size of the boards must be as small as possible. |
| **HF-2** | The position of user buttons and the signing light must be easily accessible. |
| **HF-3** | The boards must be interconnected through an accessible pin header. |
| **HF-4** | The input of the encoders and the connection of the motors must be aligned. |
| | **Communications** |
| **HC-1** | The prototype must contain a high-speed ethernet peripheral. |
| **HC-2** | The board must contain a CAN bus with an external port. |
| **HC-3** | The board must contain an UART bus with an external port. |
| **HC-4** | Optionally, the system may contain a USB port with a USB-C connector |
| | **Power** |
| **HP-1** | The system must be able to control at least two motors at the same time. |
| **HP-1.1** | The voltage of the motors may be in the range from +6 to +12 V |
| **HP-1.2** | Each motor may require currents up to 5A DC and 10A peak. |
| **HP-2** | The power input must be the same for the control and power blocks. |
| **HP-3** | There must a 5V output in each digital connector. |
| **HP-4** | The analog supply must be separated from the digital one. |
| | **Sensing** |
| **HS-1** | The system must include one encoder input per motor output. |
| **HS-2** | The board must contain at least 4 analog inputs. |
| **HS-2.1** | The analog input must be capable of sensing analog strain gauges. |
| **HS-2.2** | The analog input must be used in single or differential mode |
| **HS-3** | The power board must contain one temperature sensor. |
| **HS-4** | Each motor output must contain one current sensor to measure the supplied current. |

The behavior of a transmission line can be modeled using the quadrupole shown in Figure 2.1. Using this model, the behavior of the transmission line is completely defined by a unique parameter known as characteristic impedance ($Z_0$). The characteristic impedance is defined as the relationship between the voltage and the current along the line. The value of $Z_0$ can be calculated using the primary parameters of the line: series Resistance ($R$), Series Inductance ($L$), Parallel Capacity ($C$), and Parallel Conductance ($G$) [19]. The relation between these parameters and the characteristic impedance is shown in Expression 2.2. The value of $Z_0$ depends on the frequency and the length of the line.

$$Z_0 = \sqrt{\frac{R + j\omega L}{G + j\omega C}} \tag{2.2}$$

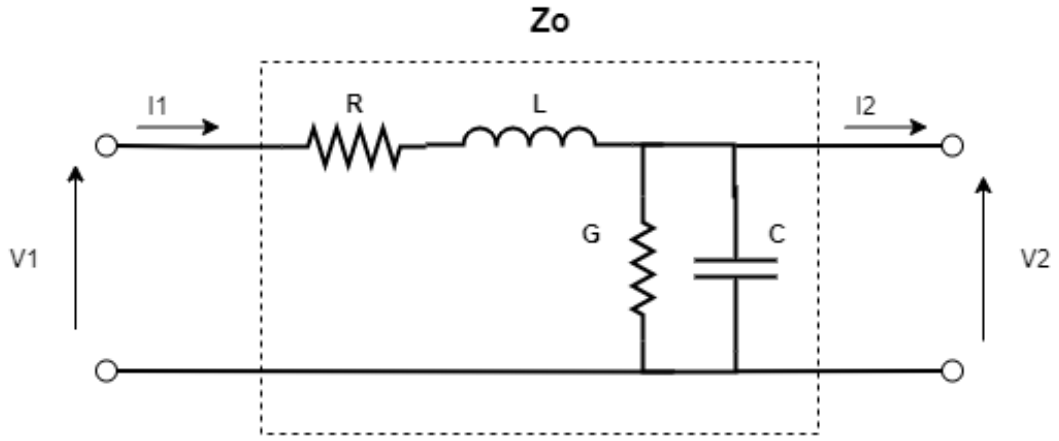



Figure 2.1: Quadrupole model of a transmission line.

Reflection losses are power losses caused by an impedance change on the transmission line. This change causes a reflection of the wave and, therefore, a second wave in the opposite direction to the main one. The magnitude and polarity of the reflected wave depend on the degree of mismatch between both impedances.

$$\Gamma = \frac{V^-}{V^+} = \frac{Z_L - Z_0}{Z_L + Z_0} \tag{2.3}$$

The relation between the incident wave ($v+$) and the reflected one ($v-$) is known as the reflection coefficient ($\Gamma$). This coefficient can be expressed in terms of the characteristic impedance $Z_0$, the mismatched impedance $Z_L$ using Expression 2.3. An adaption mismatch causes power losses because not all power is delivered to the load, instead some are reflected. These losses are commonly known as return losses and are defined by Expression 2.4.

$$RL\ [dB] = 10 \times log_{10}\left(\frac{P_{incident}}{P_{reflected}}\right) \tag{2.4}$$

### 2.3.2 Impedance matching

As mentioned above, the characteristic impedance depends on the geometry of the conductors and dielectric that shapes the environment through the wave travels. Impedance matching is the process by which the geometry of the transmission line is changed to achieve a desired characteristic impedance.

Impedance control is especially important in high-speed buses due to the potential negative impact on bit error rate, distortions, reflections, and power losses that can occur without proper impedance matching. Some of these buses have a predefined impedance that must be achieved by the transmission line to ensure proper work [17].

The typical line geometries used on a printed circuit board (commonly known as a PCB) are shown in Figure 2.2. The impedance of the traces depends on their geometry, materials, and layer arrangement of the PCB. [43].

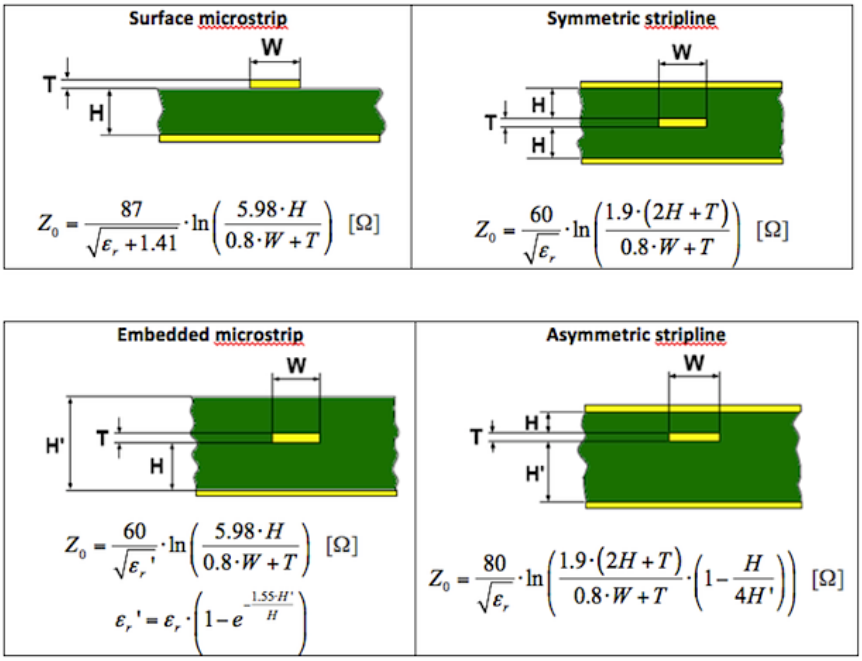



Figure 2.2: Controlled impedance configurations [46].

### 2.3.3 Digital Single-ended and differential lines

Single-ended and differential signaling are two methods for transferring information between two components [59]. As shown in Figure 2.3b, single-ended is the simplest and most widely used method of transferring information between devices because it requires only one wire per signal, with all sharing the same reference line (ground)[23]. These lines only carry two possible voltages: HIGH to identify a logic '1' or LOW to identify a logic '0' [52].

Since in single-ended signaling all signals share a common reference plane (ground), any noise introduced here would affect all lines, causing ripples in the signals. As an example, power supplies are the most dominant sources of noise in circuit boards. Furthermore, it is sensitive to the induced noise along the path.


(a) Differential pair.

(b) Single ended.

Figure 2.3: Differential and single ended lines comparison [59].

Some of these problems can be compensated using differential signaling. As shown in Figure 2.3a, differential signaling is a method that, in contrast to single-ended signaling, uses two wires to transmit each signal. Differential pairs are often used in high-speed buses where strong noise immunity is needed, as in the case of Ethernet or USB.

Each wire of the differential pair carries a voltage level with the same magnitude but with opposite polarity [59], the original signal is recovered by taking the difference between both lines (see Figure 2.4). The key idea is that any noise will be coupled in both lines with the same magnitude and, therefore, by taking the difference between the lines, the common noise would be compensated.

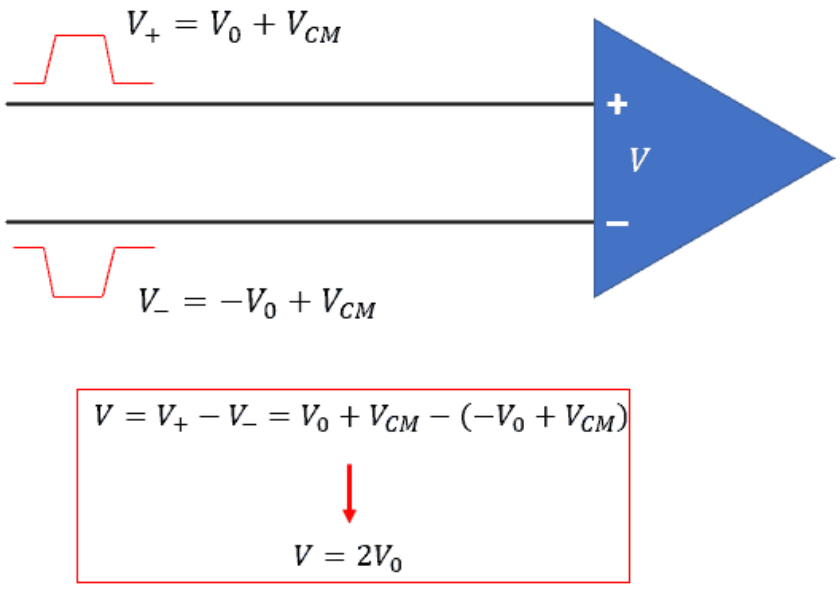



Figure 2.4: Differential signal recovery at a receiver component [59].

## 2.4 Thermal dissipation.

Joule effect is a physical phenomenon by which the passing of an electric current through an electrical conductor produces thermal energy [55]. Thermal energy generates a temperature rise that heats the conductor. Thermal conduction, also known as heat conduction, is defined as the transfer of thermal energy between two bodies with different temperatures.

In engineering, the heat flow is often expressed in terms of thermal resistance. Thermal resistance is a physical property by which an object or material resists heat flow. It is an analogy between the diffusion of heat and electric charge: as electrical resistance is associated with the conduction of electricity, thermal resistance is associated with the conduction of heat [7].

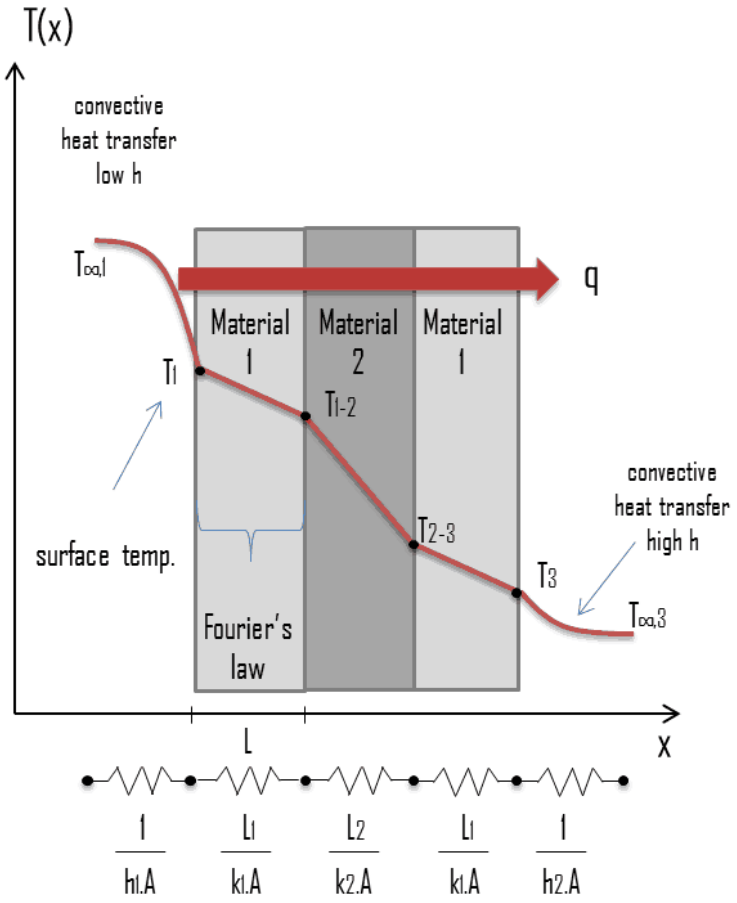



Figure 2.5: Thermal resistance analogy [7].

The heat flow through the thermal conductor of Figure 2.5 is given by Fourier's law of heat conduction in Expression 2.5.

$$Q = -KA\frac{T_2 - T_1}{L} \tag{2.5}$$

Where $Q$ is the heat flux through the plane, $K$ is the conductivity of the materials, $L$ is the thickness of the plane, and $A$ is the area of the plane.

$$R_{th} = \frac{L}{KA} \tag{2.6}$$

Now, by the definition of $R_{th}$ in Expression 2.6, the analogy to the Ohm law can be seen. As in the electrical case, thermal resistances can be combined to solve problems in which heat flows through different thermal conductors, as shown in Expression 2.7.

$$Q = \frac{T_{\infty,1} - T_{\infty,3}}{R_{total}} = \frac{\Delta T}{R_{\infty,1} + R_1 + R_2 + R_1 + T_{\infty,1}} \tag{2.7}$$

The thermal resistance is a key parameter that is currently provided by almost all manufacturers of electronic components. Manufacturers provide empirical information about the thermal resistance between the silicon (junction) and the case and the

maximum power dissipation and/or junction temperature at the junction supported by the device. This information must be taken into account when designing high-power systems to avoid thermal breakdowns.

## 2.5 Block diagram

Using the requirements listed in Table 2.1, the hardware has been divided into five functional blocks as shown in Figure 2.6. This block diagram divides the hardware project into smaller blocks that make description, design, and implementation easier. In the following sections, the different parts that make up each block and how they are made are explained in detail.

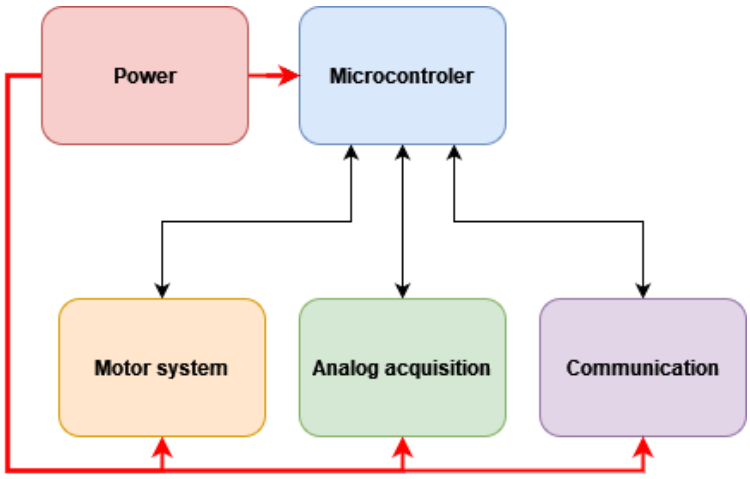



Figure 2.6: Hardware block diagram.

### 2.5.1 Control block

This block is made up of all the parts in charge of processing the input signals and the generation of digital stimuli. In summary, it is made up of a microcontroller and its associated electronics.

#### 2.5.1.1 Microcontroller selection

The choice of microcontroller depends on the required capabilities of the system. Using the requirements listed in Table 2.1, the Simplia COMPAC IMXRT module has been selected [54]. IMXRT Module is a M.2 2230 format board based on an IMXRT 1052 processor with with a Plug and Trust device to provide a root of trust at IC level. Moreover, multiple interfaces such as LCD, USB, CSI, SD, and RMII make this module suitable for different applications [54]. The i.MX RT1050 MCU runs on the Arm Cortex-M7 core at 600 MHz and its one of the first crossover microcontrollers that combine the high-performance and high level of integration of an application

processor with the ease of use and real-time functionality of a microcontroller [41]. The internal structure of the COMPAC SoM is shown in Figure 2.7.

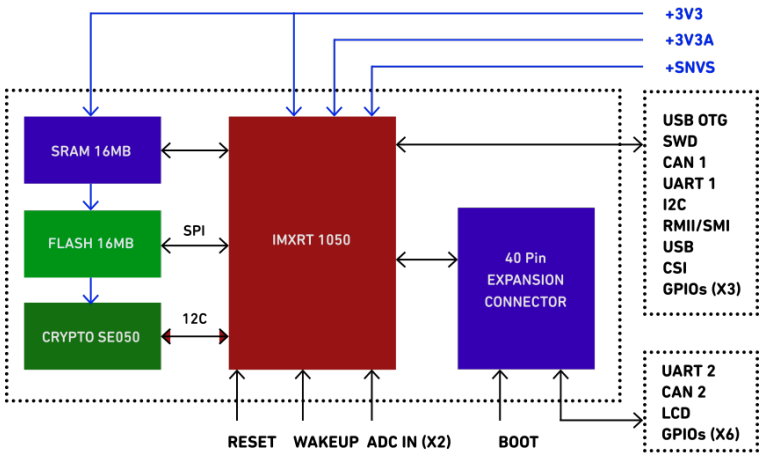



Figure 2.7: COMPAC SoM diagram [54].

This module has been selected not only for its incredible performance but also for the complete software development kit (SDK) that the manufacturer provides. This SDK provides the necessary configuration, documentation, drivers, and middleware to facilitate software development. Furthermore, it is a ready-to-use module or, in other words,it does not require extra hardware to start to work with the microcontroller.

### 2.5.2 Communication block

#### 2.5.2.1 Ethernet

Ethernet is an international standard for Local and Metropolitan Area Networks (LANs and MANs), employing CSMA/CD (carrier sense multiple access with collision detection) as the media access method and the IEEE 802.3 protocol and frame format for communications [12].

Ethernet standard lies in the lower layers of the OSI model, defining the data link and the physical layers. The data link layer, among other components, is composed of the Medium Access Control block (MAC), which functions as the interface between the processing systems (CPU, MCU, FPGA, etc.) and the physical layer.

On the other hand, the physical layer is in charge of sending the data incoming from the MAC layer through the physical networking interface as an analog signal [57]. The most common physical mediums are coaxial, twisted pair, and optical cables. In general, these tasks are performed off the processing chip by an integrated circuit known as Ethernet PHY Controller. Ethernet-capable devices have associated MII and/or RMII peripherals, which work as interfaces between the MAC and the physical layers.

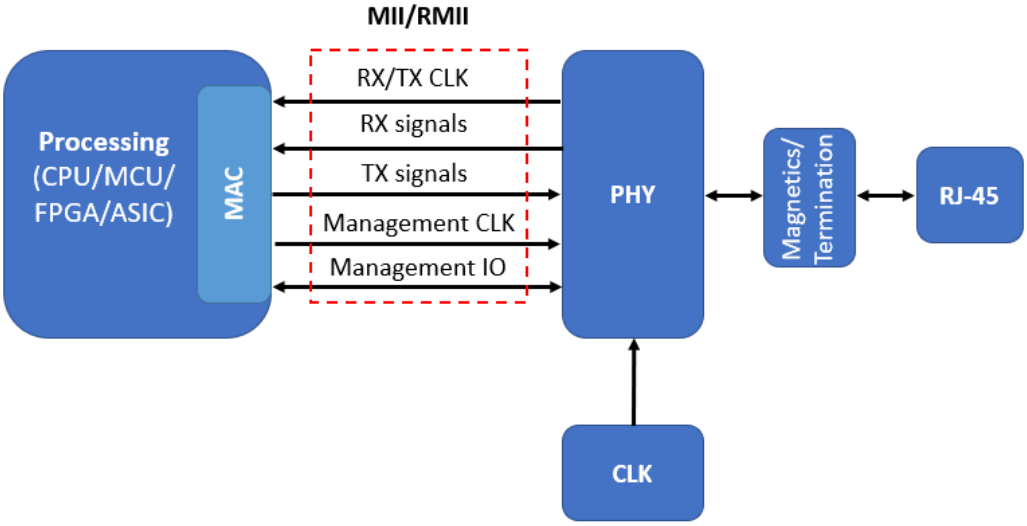



Figure 2.8: Block diagram showing connectivity in an Ethernet device [57].

MII stands for Media-Independent Interface and requires up to 18 lines to communicate each PHY controller with the MAC. Therefore, Reduced MII or RMII is a variant of MII which cuts the number of necessary lines down to 8 per PHY controller. RMII specification is capable of supporting 10 Mbps and 100 Mbps data rates. The recommended trace impedance for MII and RMII controllers is 50 Ω and it is also recommended to match the length of the traces and make them as short as possible. Figure 2.8 shows a typical connectivity diagram of an Ethernet device.

Ethernet over twisted pair is the preferred physical layer standard for network cabling applications at distances less than 100 m [1]. This medium is formed by up to 4 twisted differential pairs with a differential trace impedance of 100 Ω. The number of twisted pairs depends on the cabling category (CAT-1 to CAT-8) and, consequently, the maximum data rate.

#### 2.5.2.2 Ethernet Design

For this implementation, the LAN8720AI-CP ethernet physical transceiver (PHY) from Microchip has been used. This 24 pin Quad-Flat no-lead chip is a 10/100 ethernet transceiver compliant with IEEE802.3/802.3u with auto-negotiation and automatic polarity detection. This chip is connected to the microcontroller MAC through the ENET peripheral. The schematic has been designed according to the manufacturer's requirements shown in Figure 2.9. As an Ethernet connector, the Link-PP LPJG0846BBNL connector has been used, which includes the required coils within the same header.

In addition, electrostatic discharge protection (ESD) has been added to the Tx / Rx pairs to avoid electrical damage when using the device. Both pairs have also been impedance-matched to 100 Ω and length-matched interpair (both pairs have the same length) and intrapair (in each pair, both lines have the same length). The schematic

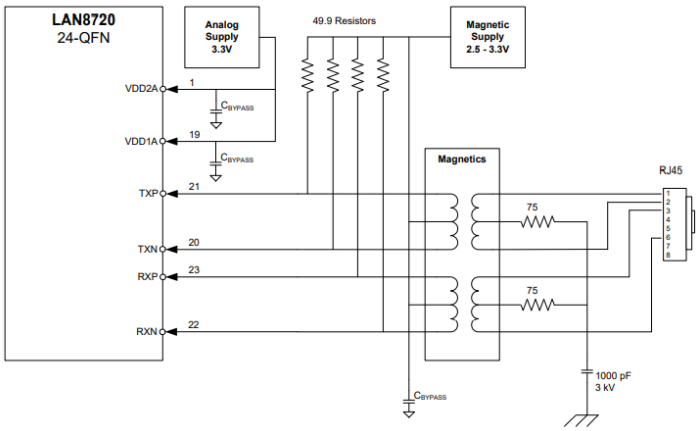



Figure 2.9: Recommended LAN8720A schematic [34]

of this module is shown in Appendix F. This block has been added to comply with the HC-1 requirement in Table 2.1.

#### 2.5.2.3 UART

UART or Universal Asynchronous Receiver/Transmitter is a serial data protocol mostly used in device-to-device communications. UART peripherals have up to two single-ended data lines that can function in simplex mode (only one data line and direction), semi-duplex (one line for both directions), or full-duplex (one line per direction) [53].

UART is a low-speed transmission protocol whose main advantage is that it is asynchronous, that is, neither of the connected devices must share the same clock signal, reducing the number of necessary lines and the complexity of the transceiver. Since they do not share a clock, the transmission speed must be agreed between the two devices. The data rate is measured in bauds or symbols per second, with the most common baud rates being 4800, 9600, 19200, 57600, and 115200 bauds.

The frame format is also quite simple; the transmission line keeps a logic '1' during the idle state. The transmission starts by pulling down the transmission line (TX) during one bit time or, what is the same, putting a '0' or start bit in the TX. Next, the binary data is transmitted starting from the least significant bit (LSB) to the most significant bit (MSB). It is noteworthy that although the frame data size can be in the range from 5 to 9 bits, the 8 bit data frames are mostly used. Optionally, the data may be followed by a parity bit that indicates its evenness or oddness of the data as a redundancy check. Lastly, the TX line must be high for at least 1 or 2 bit times (depending on the configuration) before a new transmission starts. A typical data frame diagram is shown in Figure 2.10.


Figure 2.10: UART frame [44].

#### 2.5.2.4 UART implementation

In order to comply with the HC-3 requirement in Table 2.1, an external UART header has been added to the board connected to the UART1 peripheral through a level translator to obtain a +5V tolerant port. The diagram of this module is shown in Figure 2.11. Moreover, it can be seen that the output header is connected to +5V and protected using an ESD diode, which meets the HP-3 requirement.

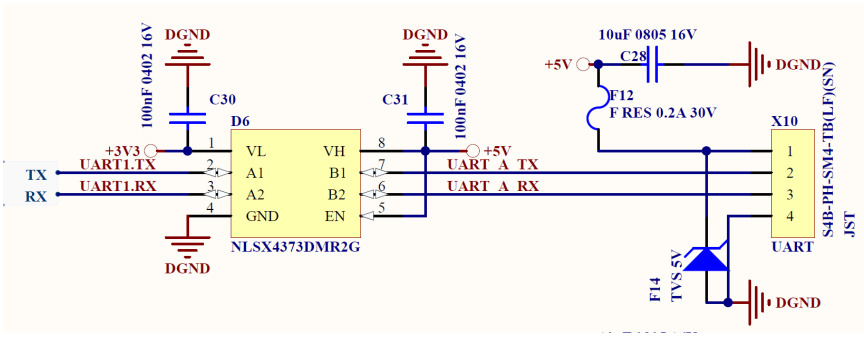



Figure 2.11: UART Schematic.

### 2.5.3 USB

USB stands for Universal Serial Bus and, as its name says, it is a serial protocol and wiring standard designed to support data exchange between a host device and a wide range of simultaneous accessible peripherals [6]. It should be noted that it has been adopted as the communication standard for numberless peripherals like keyboards, mouses, massive storage, printers, and so on, replacing legacy interfaces like RS232, parallel port, and PS/2.

This standard defines the bus topology, types of devices, data flow, and even the physical wires and connectors. The basic USB cable is made up of two power conductors and a twisted pair as a differential transmission line. In addition to that, newer versions add two additional twisted pairs to provide enhanced SuperSpeed data paths, one for the transmit path and the other for the receive path [61]. The impedance of each twisted pair is matched at 90 Ω.

As shown in Figure 2.12, there is a wide range of standard connectors, USB Type C being the last version. Type-C connectors are becoming widely used since they provide great transmission performance, are reversible, and can support up to 100 W at different voltages [58].

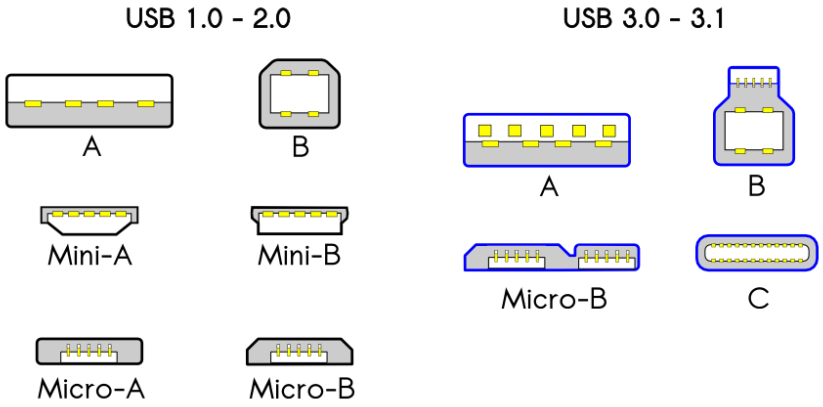



Figure 2.12: USB standard connectors [20].

Depending on the version of the specification, USB can achieve data rates from 1.5 Mbps in USB1.1 to 20 Gbps in the latest release USB4.0. One of the main features of USB is that it is designed not only to serve as a communication interface, but also to function as a power supply. According to the USB specification, USB1.1 and USB2.0 can supply up to 500 mA, while USB3.0 can provide up to 1.5 A. Some consumer electronics use only this feature to provide power input through USB connectors.

### 2.5.4 USB design

As shown in Figure 2.13, a USB C connector has been added to the USB1 peripheral (USB2.0). It only uses a pair protected by the ECMF02-4CMX8 ESD IC which incorporates a common mode filter. The USB pair has been impedance matched to 90 Ω and length matched to obtain the same length in both lines. Some unused lines have been pulled up or pulled down to avoid unwanted behavior.

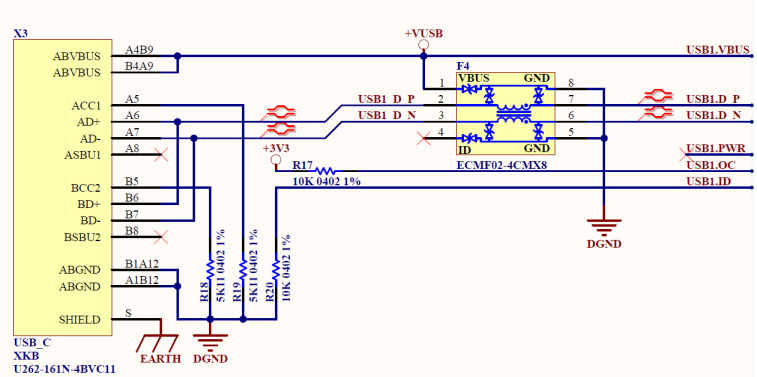



Figure 2.13: USB schematic.

### 2.5.5 CAN

The Controller Area Network bus or CAN bus is a fully centralized asynchronous communication protocol defined by the ISO-11898 standard. First introduced for automotive applications, it is also widely used in many industrial applications where efficient and robust communications are needed. Two versions are now in use: a low-speed version known as CAN 2.0A or Basic/Standard CAN, and a high-speed version known as CAN 2.0B or Extended-Frame CAN [60].

The CAN bus is formed by a two-wire differential interface that runs over a twisted pair. It uses a non-return zero (NRZ) codification scheme which ensures compact messages with minimum number of transitions and high resilience to external perturbations. The CAN bus can work with different data rates in the range of 10 Kbps to 1 Mbps, with the 20 kbps data rate compulsory in all devices. Moreover, this bus can reach up to 1000 m working at low data rates.

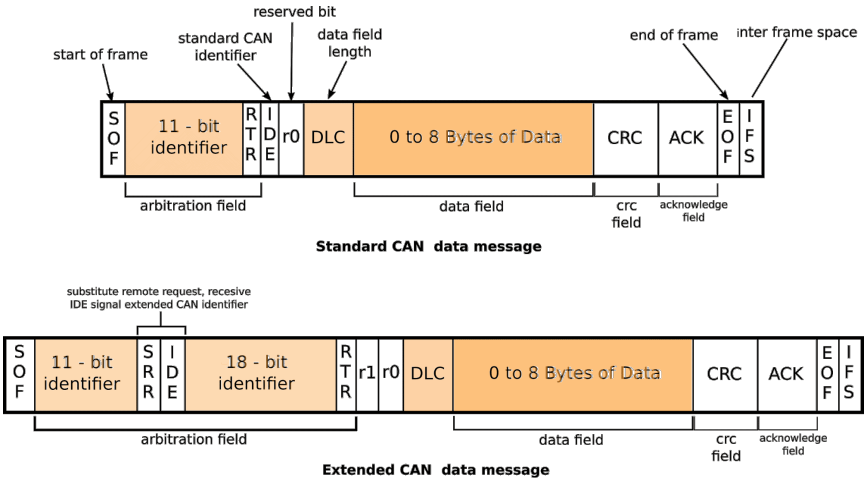



Figure 2.14: Standard and Extended frame of the CAN data message architecture [56].

The basic CAN frame is primarily composed of a Start-Of-frame block (SOF), an 11 bit message identifier, and up to 8 bytes of payload. It can also contain a CRC block, an acknowledgment slot, and other overheads as shown in Figure 2.14. ID slot is also used to encode the message priority, providing all devices with a package filtering mechanism.

The CAN bus is used in practically all modern vehicles and industrial devices, mainly due to the following benefits [11]:

- Simple: systems can communicate using only two wires, reducing errors, wiring, weight, and cost. Furthermore, CAN transceivers and controllers are simple, inexpensive, and easy to implement.

- Fully centralized: the CAN bus provides only one point of entry for all network devices.
- Robust to disturbances: the bus structure makes the system extremely robust to external interference, which is a key feature in noisy applications.
- Efficient: the CAN bus uses prioritized frames that guarantee a fast response to critical operations. Therefore, it enables the use of the bus CAN in real-time critical applications.

### 2.5.6 CAN Implementation

This block has been implemented using the NCV7351D1ER2G transceiver manufactured by ON-Semiconductor. The NCV7351 CAN transceiver is the interface between a controller area network (CAN) protocol controller and the physical bus. Furthermore, it is compatible with the ISO 11898-2 standard providing a transmission speed up to 1 Mbps. With the aim of making the bus +5V tolerant, a level translator has been added between the microcontroller (+3.3 V tolerant) and the transceiver (+5V tolerant).

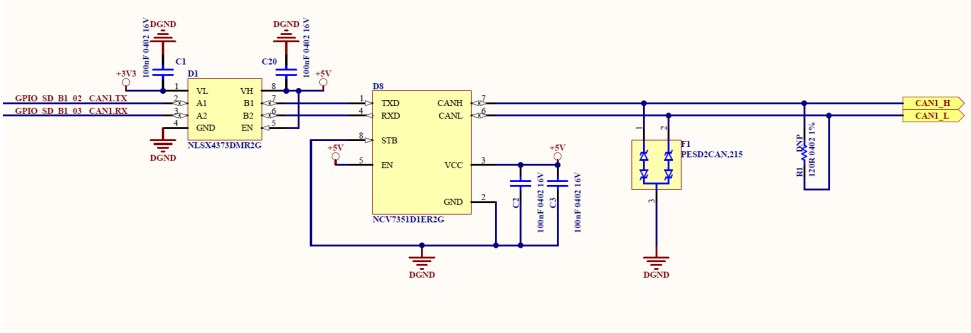



Figure 2.15: CAN schematic.

As shown in Figure 2.15, ESD protection has also been added to avoid electrical damage due to electrostatic discharges. A 120 Ω end line resistor, not mounted by default, which can be used in the case of working as an end device or in one-to-one communications.

### 2.5.7 Brushed Motor driver

#### 2.5.7.1 H-Bridge

An H-bridge is a structure composed of four switches, typically used to control the direction of current that follows through the power rail. The structure of an H-bridge is shown in Figure 2.16. The name of this structure came from the way the switches are connected.

The switches have different functions: the two switches at the top are known as the 'high side' and are responsible for connecting the DC to the power rail. The two

in the bottom are known as the 'low side' and are responsible for connecting the load to the ground [3]. As shown in Figure 2.16, the activation pattern of these switches defines the current path and therefore the direction of the motor.

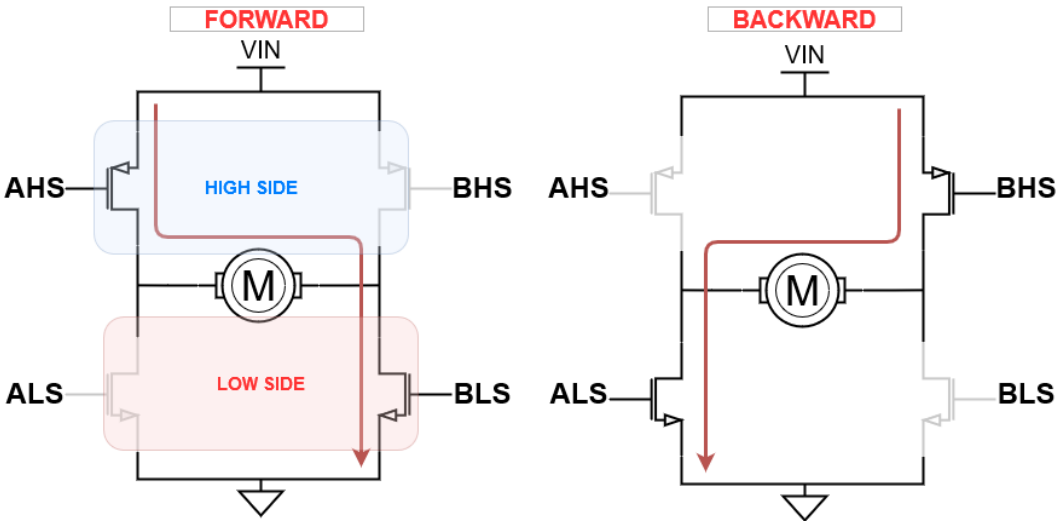



Figure 2.16: Typical H-Bridge output operating configurations.

Although there are many possible implementations, most applications use Metal Oxide Semiconductor Field Effect Transistors (also known as MOSFETs) as switches. The main advantage benefit of using MOSFETs over other types of transistors like Bipolar Junction transistors (also known as BJT) is that smaller MOSFETs can drive higher currents with lower power losses and therefore fewer amounts of heat.

MOSFETS can be P-Channel or N-Channel depending on the application and requirements. For example, in H-Bridge applications, the high side is usually composed of P-channel MOSFETs, whereas the low side is composed of N-channel MOSFETs. This structure is not arbitrary; it responds to practical reasons: to drive both N and P MOSFETs, it is necessary that the voltage between its gate and source pins be greater than a certain threshold value. Furthermore, in P-MOSFETs, the source pin must have a higher voltage than the drain one, and, by contrast, in N-MOSFETs, the source pin must have a lower voltage. In the H-Bridge structure, the P and N MOSFET sources are connected to the input voltage and to the ground, respectively.

When selecting a MOSFET device for high-power applications, there are several factors to consider, such as size, form factor, power rating, working voltage and on-resistance. On-resistance ($R_{on}$)is defined as the equivalent series resistance present between the source and the drain when the MOSFET is operating in the active region or when the device gate is fully charged [3]. This resistance value becomes especially important, since it defines the amount of power dissipated by the device. The amount of power dissipated in a MOSFET transistor is given by Expression 2.8, where $i$ is

the current flowing through the transistor. Eventually, excess power will destroy the device.

$$P_d = R_{on} \times i^2(W) \tag{2.8}$$

One of the main disadvantages of P-MOSFETS compared to N-MOSFET is that the electron mobility inside a N-MOSFET transistor is around twice that of P-MOSFET, and therefore, with the same geometry, N-MOSFET has almost half the resistance compared to P-MOSFET transistors when working with the same operation conditions. This fact is especially important because the power dissipated by a P-MOSFET becomes twice. One solution is to use N-MOSFETS on both sides (high and low). However, special controllers would need to bootstrapping the device to maintain the correct voltage between the source and gate pins of the high-side MOSFETs.

Today, there are many inexpensive devices specially designed to drive N-MOSFET bridges with all-in-package solutions that can be easily implemented in projects. In the following chapters it will be discussed these solutions more deeply.

#### 2.5.7.2 H-Brigde design

For this project, two different strategies have been adopted when implementing the H-Bridge. On the one hand, one board has been designed using the MC33HB2001 monolithic H-Bridge Power IC. This device can control inductive loads with current peaks up to 10A. Furthermore, the HB2001 is enhanced with SPI communications that provide configuration and diagnostic capabilities. In addition, this device provides an analog feedback proportional to the current supplied to the load that allows to meet the HS-4 requirement in Table 2.1.

Since this is a monolithic solution, it requires fewer components and therefore a smaller area and a lower price. However, it is limited by its thermal capabilities. According to the HB2001 datasheet, this device has a thermal resistance $R_{\Theta JA}$ of 75.7 $°C/W$, a maximum junction temperature $T_{J(max)}$ of 150 $°C$, and each MOSFET transistor has a maximum drain-to-source resistance $R_{DS(on)}$ of 125 $m\Omega$. Assuming an ambient temperature $T_A$ of 25 $°C$, the maximum dissipated power $W_{max}$ can be calculated as shown in Expression 2.9.

$$W_{max} = \frac{(T_{J(max)} - T_A)}{R_{\Theta JA}} = \frac{(150 - 25)}{75.7} = 1.65W \tag{2.9}$$

Assuming that almost all the power is dissipated by the MOSFET transistors and that there are always at least two transistors on, the maximum worst-case current supplied to the load can be calculated as shown in Expression 2.10. This means that, in the worst case, this IC can provide a constant current of 2.57 A.

$$I_{max} = \sqrt{\frac{W_{max}}{2 \times R_{DS(on)}}} = \sqrt{\frac{1.65}{2.125}} = 2.57A \tag{2.10}$$

With the aim of overcoming these limitations, a second board has been designed using a fully custom design. For this design, the 85V full-bridge MOSFET driver MIC4606 manufactured by Microchip has been used, which features adaptive dead time and shoot-through protection. This driver is capable of controlling four N-MOSFET transistors using just two PWM inputs, one for each side of the H-Bridge.

The H-Bridge is formed by 4 SISS50DN N-MOSFET transistors manufactured by Vishay. According to the datasheet, each transistor has a maximum $R_{DS(on)}$ of 4.1 $m\Omega$, a thermal resistance $R_{\Theta JA}$ of 25 $°C/W$, and a maximum junction temperature $T_{J(max)}$ of 150 $°C$. Using Expression 2.9 and assuming an ambient temperature $T_A$ of 25 $°C$, the maximum dissipated power and, therefore, the maximum supplied current can be calculated as shown in Expression 2.11. Notice that, unlike the monolithic design, for these calculations it is assumed that in the worst case only one transistor is switched on. This happens because in the fully custom design, each transistor has its own body; however, in the monolithic case, all transistors share the same body.

$$W_{max} = \frac{(T_{J(max)} - T_A)}{R_{\Theta JA}} = \frac{(150 - 25)}{25} = 5W \tag{2.11}$$

$$I_{max} = \sqrt{\frac{W_{max}}{R_{DS(on)}}} = \sqrt{\frac{5}{0.0041}} = 34A \tag{2.12}$$

It is evident that in the second case the maximum power increases more than 3 times. Nevertheless, this design requires more components, a more complex layout, and hence higher cost and a much larger PCB area. This design requires flyback diodes to protect to eliminate the sudden voltage spike caused by the inductive load when the transistor is switched, ESD diodes to protect the transistor's gates, and extra components required by the bridge driver.

Furthermore, a current sensing circuit to meet the HS-4 requirement shown in Table 2.1. This circuit is made of the INA281A1 current sense amplifier manufactured by Texas Instruments and two parallel sensing resistors of 10 $m\Omega$. Given that the current amplifier is powered by the +3.3V power rail and its gain is 20 $V/V$, the maximum sensed current can be calculated as shown in Expression 2.13.

$$V_{in(max)} = \frac{V_{V_{cc}}}{G} = \frac{3.3}{20} = 165mV \tag{2.13}$$

$$I_{max} = \frac{V_{in(max)}}{R_{sense(par)}} = \frac{165mV}{5m\Omega} = 33A \tag{2.14}$$

However, it is necessary to calculate the maximum current supported by the resistors that, according to the datasheet, are capable of dissipating up to 1 W. The maximum current is calculated as shown in Expression 2.15. The two sensing resistors have been configured in parallel to duplicate the maximum power dissipated by them, giving a maximum sensing current of 20 A.

$$I_{max} = = \sqrt{\frac{W_{max}}{R_{sense}}} = \sqrt{\frac{1}{0.01}} = 10A \tag{2.15}$$

In both designs, monolithic and fully custom, a low-pass filter has been added to the current feedback path to measure only the average current supplied to the load. This filter has been designed using a second-order Sallen Key structure [8] with cutoff frequency $f_c = 1224Hz$, quality factor $Q = 0.5$, gain $G = 1$ and damping ratio $\zeta = 1$.

The schematics of both boards are shown in Appendices G and H. In both cases, a fan has been added a fan above the logic to reduce the probability of thermal damage.

### 2.5.8 Analog acquisition

Despite its digital nature, most electronic devices must interact with a pure analog world, making it necessary the existence of special circuits which work as an interface between the analog world and the digital processing units. An analog-to-digital converter, also known as ADC, is an electronic device capable of converting an analog input signal to a digital signal value that represents the value of the analog signal compared to a reference voltage. These circuits can be embedded into complex processing systems (microcontrollers, FPGAs, etc.) or as individual integrated circuits.

The analog input to be measured can be single-ended or differential, but always must be in the range of the reference voltage ($V_{ref}$) or FSR. The $FSR$ or full-scale input range is divided into a number of subranges or steps, each one with an assigned binary code. Resolution is defined as the number of bits used to encode each step. With $n$ bits, an ADC can encode up to $2^n - 1$ steps. For example, a 12 bit ADC can be encoded in $2^{12} - 1 = 4095$ steps. The minimum voltage that guarantees a change in the output code is called the least significant bit (LSB) voltage. Therefore, the resolution determines the precision of a measurement.

The sample rate is defined as the number of samples that can be taken per unit of time. According to the Nyquist theorem, an analog input can be reconstructed without information loss as long as it is sampled at a sample rate greater than or equal to twice its maximum frequency. This is a key feature to consider to avoid side effects such as aliasing.

#### 2.5.8.1 Analog acquisition implementation

For this block, the MCP3564T ADC manufactured by Microchip has been used. The MCP35614 device is an 8-channel, 24-bit Delta-Sigma analog-to-digital converter (ADC) with a programmable data rate of up to 153.6 ksps. A filtered low-noise +3.3V power rail has been used as the voltage reference for the ADC. The configuration of the eight analog channels is controlled by four analog multiplexers, providing three different configurations: differential input, single-ended input, and strain gauge input.

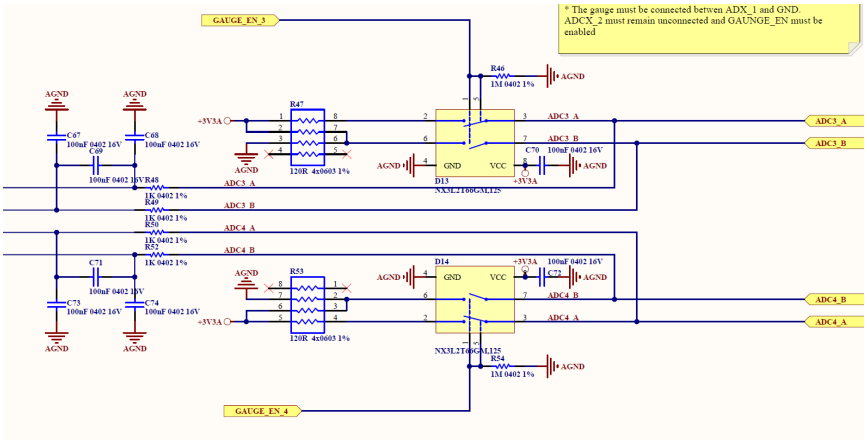



Figure 2.17: Analog input schematic.

The selected multiplexer is the NX3L2T66GM manufactured by NXP with a series resistance of 0.5 $m\Omega$. As shown in Figure 2.17, the multiplexer controls the connection and disconnection of the 120 Ω Wiston bridge used to linearize the input of the strain gauge. This bridge is made of a monolithic resistor array to achieve the same value, tolerance, and temperature shift in all resistors.

All inputs are connected to the ADC through a low-pass filter that acts as an anti-aliasing filter with a cutoff frequency $f_c = 1.6KHz$. To avoid filter gain dismatches with the differential input configuration, a common mode capacitor has also been used between every two consecutive channels. The complete schematic diagram for this block is shown in Appendix F.

### 2.5.9 Power supply

The power rail block is one of the most critical parts in designing an electronic device. A simple hardware design may contain many components with different supply requirements, and consequently, it must use different power rails at different voltages with loads ranging from a few milliamperes to hundreds of amperes. The design of these power rails must meet requirements such as power efficiency, PCB space and layout, transient behavior, and cost.

In summary, a power supply can be defined as a component or group of components capable of generating a regulated power output (voltage and/or current) from an unregulated power input. It can also provide additional features such as current and heat measurement and overload protection. There are many different internal structures for these devices depending on the characteristics of the input and output sources, such as the maximum voltage and current and the type of source (alternating current or AC and direct current or DC).

This section will focus on DC/DC power supplies which, as implied by the mane, use an unregulated direct current input (DC) to generate a regulated DC output.

In this category of power supplies, there are two main internal structures: switching mode power supplies (SMPS) and linear regulators. To make a good choice, it is necessary to know the merits and drawbacks of each structure. Therefore, in the following sections, a brief description of each will be given.

#### 2.5.9.1 Linear regulators

A linear regulator is an electronic device capable of generating a regulated voltage output by varying an active series load. In other words, it can be seen as a series resistor connected between the input and the output, whose resistance changes to maintain the desired voltage at the output. This load regulation is done using a feedback control system, as shown in Figure 2.18.

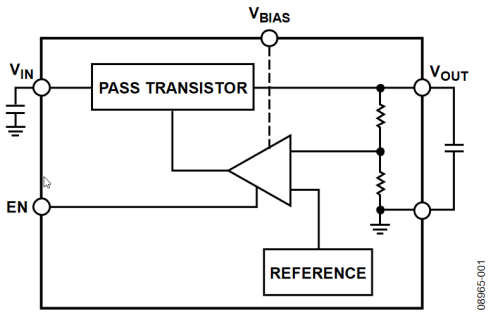



Figure 2.18: Basic structure of a linear regulator [29].

These regulators are simple, inexpensive, and do not need many extra electronics (usually only a couple of decoupling capacitors). Furthermore, linear regulators have a low noise figure, high accuracy, and high power supply rejection ratio (PSRR), providing, as a result, a highly regulated output voltage with low noise. Therefore, these features are highly desirable, especially in analog applications.

However, they are not suitable for all applications for some reasons: firstly, they are only step-down regulators, which means that they can only generate a lower voltage than the input. Moreover, there must be a minimum voltage difference between the input and output, which is also known as dropout voltage ($V_{DO}$). However, its biggest drawback is its low power efficiency.

As shown in Expression 2.17, power efficiency can be defined as the relationship between the power provided at the input and the power delivered to the load. Since the output current is virtually the same as the input (LR acts as a load), the power efficiency is defined by the voltage ratio between the output and the input. To illustrate the problem, imagine that a linear regulator is used to regulate from +12V to +5V, giving an approximate power efficiency of 40%, which means that the remaining 60% are dissipated by the device as heat.

$$P_{diss} = (V_{in} - V_{out}) \times I_{out} = V_{DO} \times I_{out} \tag{2.16}$$

$$\eta = \frac{V_{out} \times I_{out}}{V_{in} \times (I_{out} + I_Q)} \times 100 \approx \frac{V_{out}}{V_{in}}\% \tag{2.17}$$

#### 2.5.9.2 Switching regulators

A Switching Mode Power Supply (SMPS) is a circuit that uses a power switch, an inductor, a diode, and an output capacitance to regulate the output voltage. The power switch transforms the input into a pulsed voltage that is smoothed using an output capacitor. The power switch, typically a Field Effect Transistor (FET), is switched by a switching controller that monitors the output in a feedback control loop.

Compared to linear regulators, SMPS regulators provide high power efficiency (higher than 90%) and low power dissipation, resulting in less heat, regardless of the relationship between input and output voltages. However, its switching nature generates undesired noise from the voltage ripple across the capacitor. Depending on the switching frequency, the selected components, and the applications,the generated noise would be a serious problem and difficult to fix. Therefore, these types of power supply are the main source of noise in modern electronics.

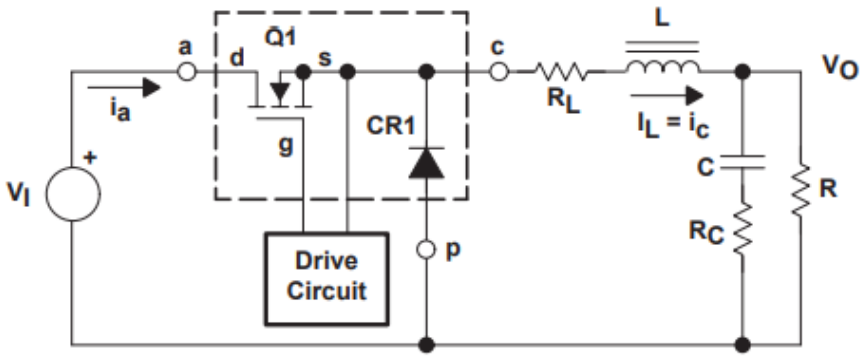



Figure 2.19: Buck Power Stage Schematic [21].

The main components of an SMPS circuit can be rearranged to form a step-down converter (Buck), a step-up converter (Boost), or an inverter Buck-Boost converter. However, in this section, we will focus on the step-down topology shown in Figure 2.19. The working principle is simple, during the time that the switch is closed ($T_{ON}$), the output capacitor (C) will be charged through the inductor (L) and, since the current flowing through the inductor can suddenly increase, the capacitor will be slowly charged maintaining a voltage across it lower than the input voltage; On the other hand, when the switch is opened ($T_{off}$), the diode starts conducting creating a current loop, allowing the inductor to inject energy to maintain the voltage across the

capacitor. As a result, the output voltage is given by the duty cycle (the relationship between $T_{ON}$ and the switching period $T_s$) and the input voltage.

### 2.5.10 Power supply design

According to the system requirements and the previous sections, the board has two possible power inputs: a +12V power input from the motor board and a +5V power supply from the USB connector. Furthermore, the system must be able to generate three power rails of +5V, +3.3V, and +3.3V with low noise. These power rails must be able to meet the power requirements shown in Table 2.2. Thus, both input power supplies must be able to supply 6.32 W to the system, which means that each supply must be able to supply the currents shown in Table 2.2. This table was calculated using the worst-case scenario.

Table 2.2: Power Requirements

| Voltage | Current | Power |
|---|---|---|
| **Required Power Rails** | | |
| +5V | 900mA | 4.5W |
| +3.3V | 300mA | 1W |
| +3.3 Analog | 400mA | 1.32W |
| | | 6.32W |
| **Required Power Inputs** | | |
| +12V | 526mA | 6.32W |
| +5V | 1264mA | 6.32W |

It should be noted that the system has been designed to use only one power source at a time, which means that if both sources are simultaneously connected, only one will supply power to the load, whereas the other is automatically disconnected. This is done using the structure shown in Figure 2.20. If the +12V supply is disconnected and the +5V supply is connected, the V2 p-channel MOSFET is active, allowing the current to flow toward the system. However, if both supplies are connected, the gate-to-source voltage across the transistor is greater than 0V, which means that V2 will be in cutoff mode. These structures are also known as load switches. The V2 transistor has been carefully selected to support maximum current with low power loss (low conduction resistance $R_{on}$).

When the +12V supply is connected, the +5V power rail is generated using the TPS5622 buck converter, manufactured by Texas Instruments, capable of supplying up to 2A and with an input voltage range from +4.3V to +17V. This IC has also been used to generate the +3.3V power rail from the +5V power rail. Both power stages have been designed using the WEBENCH Power Desing Tool provided by Texas Instruments. Furthermore, the basic design generated by the tool has been modified to use cheaper and available components with the same electrical characteristics.

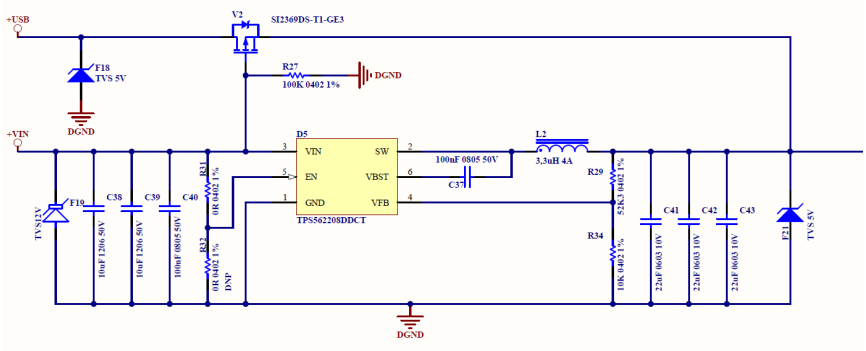



Figure 2.20: Input power schematic.

These power supplies have been designed to provide up to 2A which, according to the requirements shown in Table 2.2, is enough to provide the desired power.

As stated in the previous sections, switching power supplies introduce unwanted high-frequency noise into the system. This noise becomes especially important when switching power supplies are used to power sensitive analog electronics such as ADCs. For this reason, the AP7343 linear regulator, manufactured by Diodes Incorporated, has been added, capable of providing 400 mA to the analog block. The full schematic of this block is shown in Appendix F.

## 2.6 Printed Circuit boards

A printed circuit board, also known as PCB, is a physical support for all different components that shape the electronic circuit. It is made of mainly a base flat material, solid or flexible, on whose faces the components that form the circuit are placed. The interconnection between the components is made with metallic traces, typically copper. The trace patterns are transferred using a mask and engraved on the copper by either physical processes (such as etching) or chemical processes (such as electrodeposition).

Basic printed circuit boards consist of two or more electrical layers separated by a dielectric insulator, commonly called prepreg, forming a stack. In the middle of the stack there is another thicker layer of insulator known as the core, whose function is to provide rigidity to the board. A PCB may have two to more than ten electrical layers, but always a pair number. External layers are coated with a thin resin layer known as solder mask, whose function is to protect copper and avoid short circuits during the soldering process. The electrical layers are interconnected using small plated holes known as vias.

The dielectric layers deeply affect the performance of the circuit board, especially in high-frequency applications. Dielectrics are defined by their thermal, electrical, magnetic, chemical, and mechanical characteristics. Fiberglass, Teflon, and polyimide are often used as dielectric materials for PCBs. FR-4, which is made of fiberglass

and resin, is the most common material used in standard PCBs due to its performance/price ratio.

Design rules (such as trace size and clearance), as well as layer stacks, are completely defined by the fabrication process and the technology applied by the manufacturer. It is important to keep this fact in mind since not all manufacturers have the same capabilities and materials to build the boards.

### 2.6.1 Board layout

The entire system has been divided into three boards according to the requirement HF-1 in Table 2.1. As a result, a stack-like structure is obtained that allows the division between the control block and the power electronics (requirement HF-1.1).Therefore, new types of motor or actuators can be used simply by changing the power board.

All boards have been designed with the same form factor, a rectangular 60x80 $mm^2$ shape with rounded corners and holes in every corner. The size is the smaller size that allows positioning all external connectors in the control board (requirement HF-1.2). The resulting structure is shown in Figures 2.21 and 2.22. The boards also contain three fiducial marks in the upper and lower layers that allow SMT placement equipment to accurately locate and place the components.

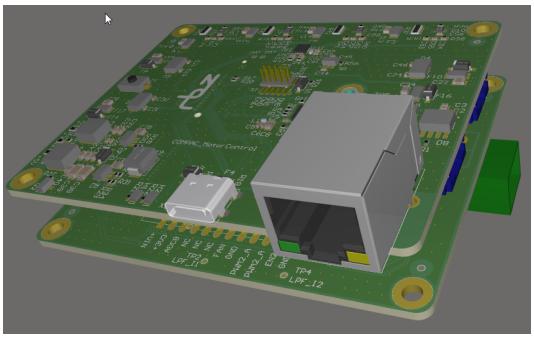


Figure 2.21: Board 3D stack top view.

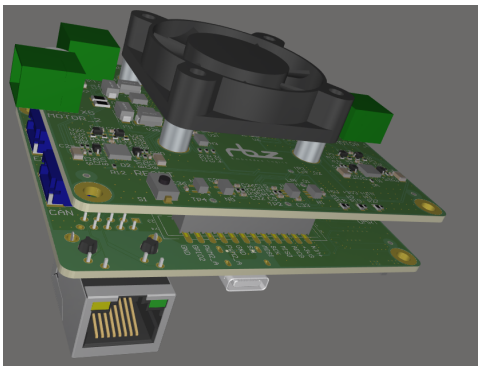


Figure 2.22: Board 3D stack bottom view.

The printed circuits have been fabricated by Eurocircuits and assembled by RBZ Embedded Logics. The first step in the assembly process requires the exposed pads to be coated with solder paste using a metallic pattern mask known as a stencil.

Then, a pick-and-place machine places the components in their correct positions. These machines can place more than 104,000 components per hour. Lastly, all components are soldered at the same time using reflow soldering. This technique consists in introducing the board into an oven, which raises the board temperature to the melting point of the solder paste. The solder paste reflows creating permanent

solder joints. Reflow soldering is suitable for soldering thousands of components in a short period of time.

#### 2.6.1.1 Control board

This board holds the main blocks of the control system: control, acquisition, power, and communications. It has been designed using the four-layer stack described in Figure 2.23. This stack is just one of the multiple stack-up options offered by the PCB manufacturer Eurocircuits. Therefore, this stack-up has been selected as a trade-off between cost and fabrication capabilities: it offers impedance control with a space between layers small enough to have good impedance matching without using thicker tracks, which means a smaller area and a higher degree of integration.

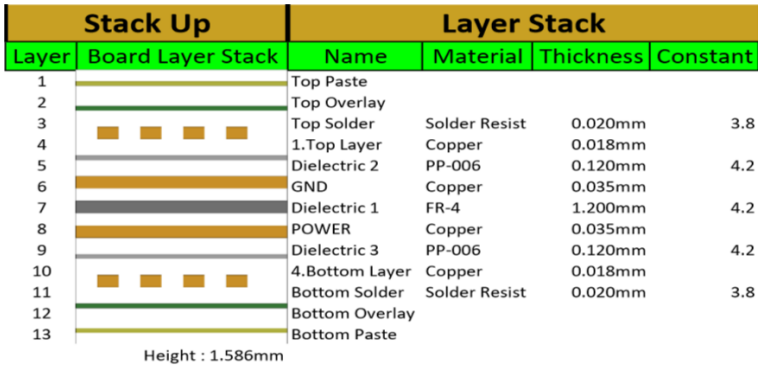



Figure 2.23: Control board stack-up.

There are two controlled impedances: a 100 Ω differential for Ethernet pairs and a 90 Ω differential for the USB pair. The size and spacing of the tracks are summarized in Table 2.3. These values were calculated using an impedance calculator provided by Eurocircuits for its stackups.

Table 2.3: Impedance control +/- 10% differential pairs.

| Layer | Impedance | Track width | Dif. Pair Space | Reference Layer |
|---|---|---|---|---|
| 1 AND 4 | 100 Ω | 0.150mm | 0.200mm | 2 AND 3 |
| 1 AND 4 | 90 Ω | 0.185mm | 0.200 | 2 AND 3 |

As shown in Figure 2.25, the board is divided into two zones: digital and analog. Both zones are joined by a single point located next to the analog linear regulator. The purpose of this structure is to avoid digital noise from coupling into the analog supply. The analog connectors are also located at the top of the board. Figure 2.24 shows the power plane division, it can be seen how the different power rails are distributed across the board.

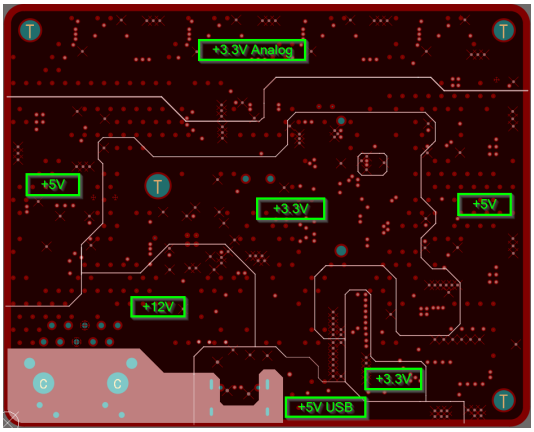



Figure 2.24: Control board power plane division.

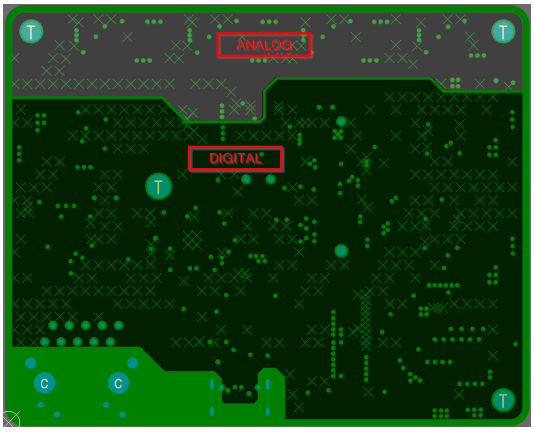



Figure 2.25: Control board ground plane division

All components are situated in both the top and bottom layers. As shown in Figure 2.27, on the top layer 4 analog inputs are located on the top, one encoder connector and the UART port on the left, and the other encoder connector and the CAN port on the right. At the bottom there is a 24 pin header that works as a connector between the control board and the motor board. The signals assigned to this port are shown in Figure 2.26. The microcontroller SoM is located in the center of the board.

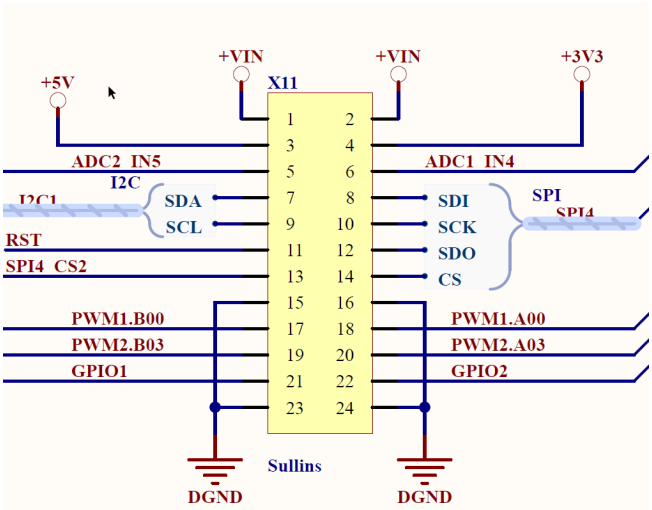



Figure 2.26: Control board connector.

On the bottom layer, most of the components are located, as well as the Ethernet and USB connectors. The power supply block has been routed following the manufacturer's instructions carefully to avoid thermal or electrical failures. At the top are situated the analog components aligned with its respective connectors in the other layer. The earth net from the USB and Ethernet connector has been conscientiously isolated from the ground of the system, removing all copper in the surrounding areas.

The Ethernet lines that come from the microcontroller have been routed following the shortest path and trying to maintain the same length in all lines. In Figure 2.28 the Ethernet and USB differential pairs that have been matched by impedance and length are highlighted. In this layer, there is also a 10 pin header that works as a programmer port. The resulting 3D model is shown in Figures 2.29 and 2.30.

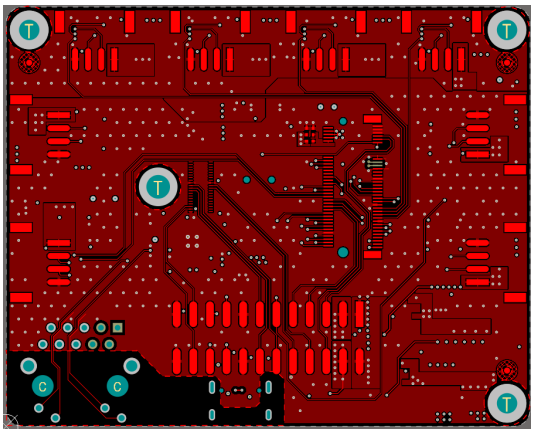


Figure 2.27: Control board top layer.

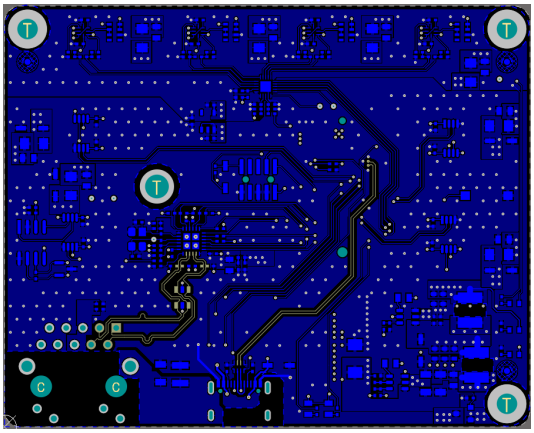


Figure 2.28: Control board bottom layer.

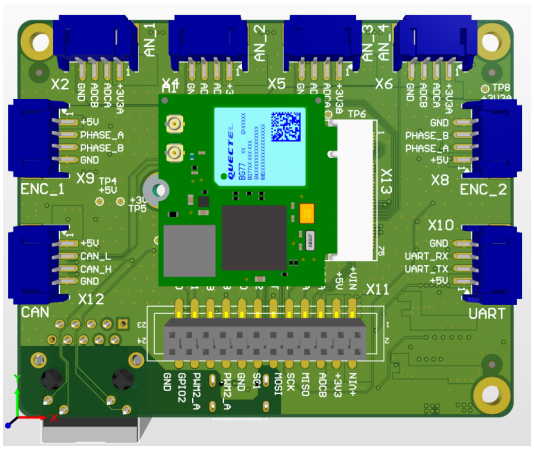



Figure 2.29: Control board top layer 3D.

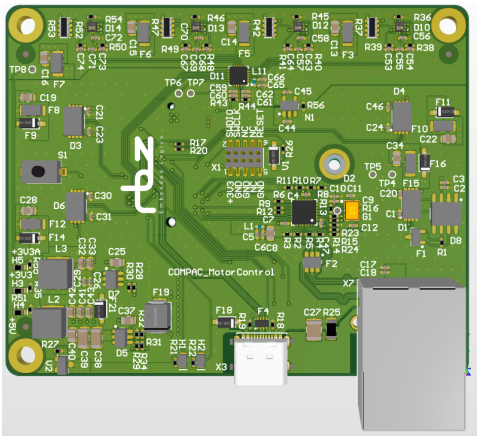


Figure 2.30: Control board bottom layer3D.

As shown in Figure 2.31, the board has been panelized in a 3x2 panel. These kinds of panel are often used to ease the assembly process carried out by automatic machines, also known as pick-and-place. Both columns are joined by a piece of PCB drilled with small holes known as 'mouse bites'. These 'mouse bites' create a weak junction that allows the board to be separated. External frames are removed using a process called V-Scoring or V-Cutting, which consists of cutting a 'v' groove on the top and bottom of a circuit board while leaving a minimum amount of material in

place to hold the boards together.

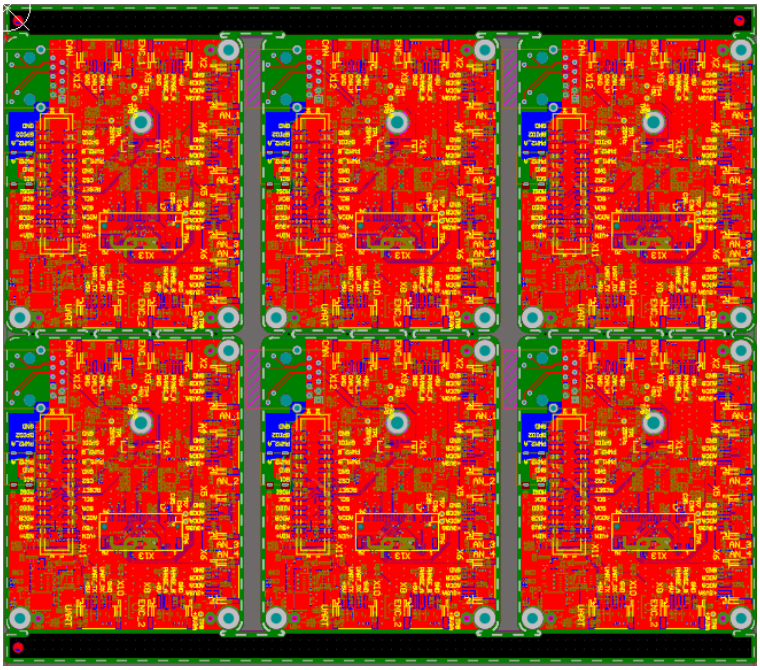


Figure 2.31: Control board panel.

#### 2.6.1.2 H-bridge MIC4606

Board Stack Report

| Stack Up | | Layer Stack | | | |
|---|---|---|---|---|---|
| Layer | Board Layer Stack | Name | Material | Thickness | Constant |
| 1 | | Top Paste | | | |
| 2 | | Top Overlay | | | |
| 3 | | Top Solder | Solder Resist | 0.010mm | 3,5 |
| 4 | | Top Layer | Copper | 0.035mm | |
| 5 | | Dielectric 1 | FR-4 | 1.550mm | 4,8 |
| 6 | | Bottom Layer | Copper | 0.035mm | |
| 7 | | Bottom Solder | Solder Resist | 0.010mm | 3,5 |
| 8 | | Bottom Overlay | | | |
| 9 | | Bottom Paste | | | |

Height : 1.640mm

Figure 2.32: Motor boards stackup.

The MIC4606 board contains the implementation of the full custom H-Bridge, and the HB2001 board contains the monolithic IC design. In contrast to the control board, for these boards, a two-layer stackup was selected manufactured by Eurocircuits. The complexity, number of components, and absence of controlled impedance allow the use of a cheaper and lower cost structure, which is shown in Figure 2.32. Since there are no internal power planes, the power rails have been routed in the top and bottom

layers.

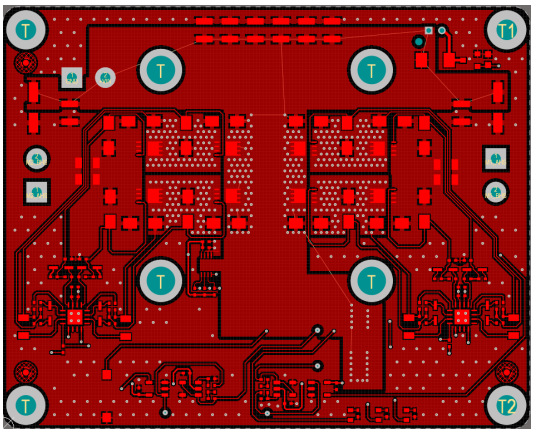


Figure 2.33: Motor MIC4606 board top layer.

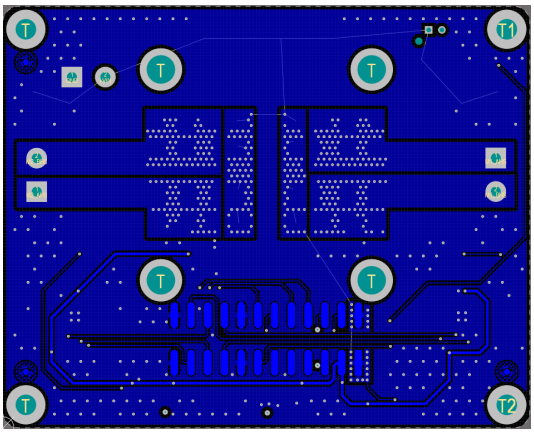


Figure 2.34: Motor MIC4606 board bottom layer.

As shown in Figures 2.33 and 2.35, all components have been located in the top layer, leaving the bottom layer to connect the power rails, the high-power outputs, and the board connector (see Figures 2.34 and 2.36). On the top is the input source connector and on the sides are the motor connectors. These connectors, as well as all other high-power components, have been connected to copper without thermal relief. Although it may cause problems during the assembly process, it was done to allow a greater amount of current to flow through the high-power components.

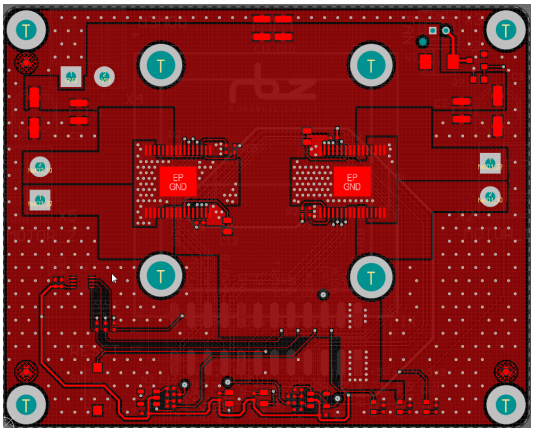


Figure 2.35: Motor HB2001 board top layer.

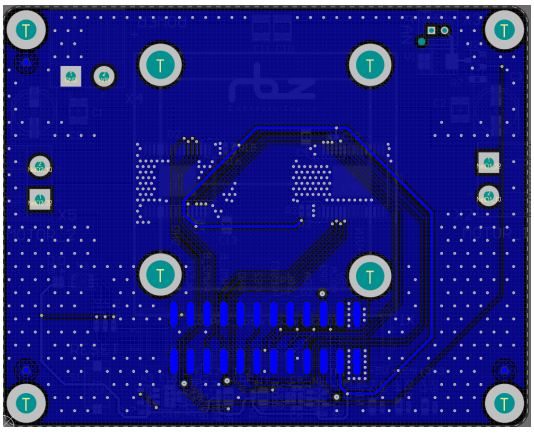


Figure 2.36: Motor HB2001 board bottom layer.

In the center of both boards, there are two H-bridges with a fan above them to avoid heating problems. The transistors and the HB2001 IC were interconnected by large copper planes on both layers. These planes are also connected through a dense

array of vias to ease heat dissipation. In addition to the H-Bridge, these boards also contain power LED indicators and a reset button on their top layer, in addition to an I2C temperature sensor, which meets the requirements of HF-2 and HS-3 in Table 2.1.

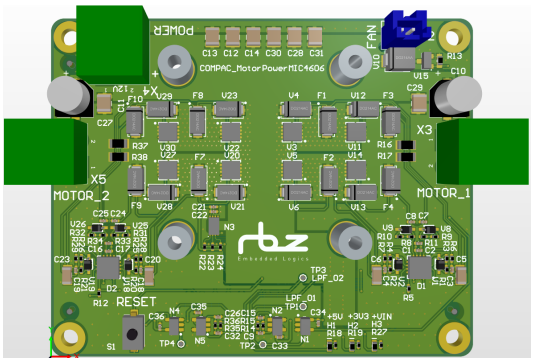



Figure 2.37: Motor MIC4606 3D model.

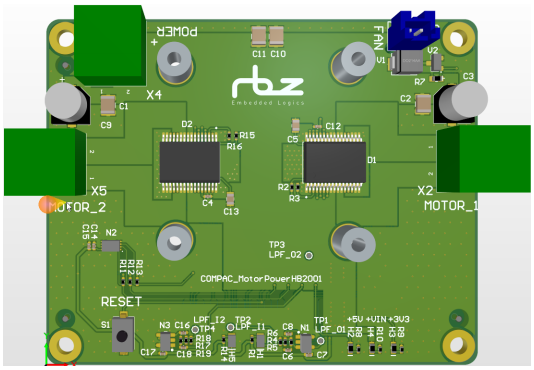



Figure 2.38: Motor HB2001 3D model.

The 3D models of both boards are shown in Figures 2.37 and 2.38. These boards have been panelized together in a 2x3 array (3 MIC4606 boards on the left side and 3 boards HB2001 on the right size). These boards are joined by a small piece of PCB drilled with multiple holes. It is worth stating that these boards can be panelized together because both share the same stackup and, therefore, share the same fabrication process. The result panel is shown in Figure 2.39.

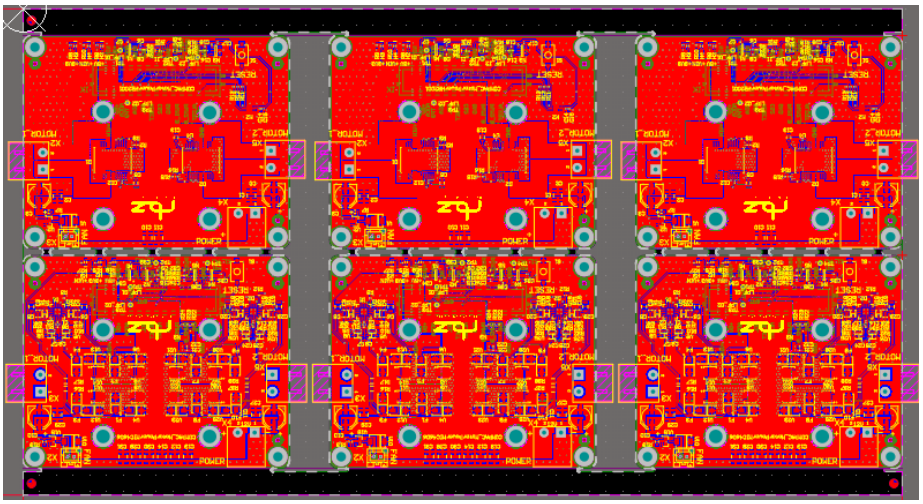


Figure 2.39: Motor boards panel.

# Chapter 3

# Software design

## 3.1 Introduction

Software is defined as a set of coded instructions for the automatic performance of the desired tasks. In an embedded system, both hardware and software are closely related to each other. Embedded software is designed, implemented, and compiled to run on a specific target hardware, typically with reduced resources and capabilities. However, this does not mean that embedded systems are slow; on the contrary, these systems are in charge of running critical tasks, with highly restrictive time constraints, also known as hard Real-Time, in which any missed deadline results in a loss of life or property.

If this were not enough, these systems must be able to operate for long periods of time without human intervention. They are called embedded systems because they are typically integrated as part of a high-level system. As a result, embedded software must be reliable and capable of detecting, reporting, and recovering errors in real time with any kind of external intervention. Software design must take care of all these requirements and constraints and take advantage of all hardware capabilities to obtain a high-performance system. As an example of the importance of embedded software design, the case of sudden unexpected acceleration (SUA) in Toyota cars can be mentioned. A pour design plagued by software bugs and more than half a dozen death tasks caused 89 deaths and 57 injuries only in the first half of the year 2000 [24]. This is only one case, but the history is full of countless others with hundreds of lives and millionaire losses.

In the following sections, the requirements and modeling of the behavior of the system will be described. Furthermore, the design, implementation, and integration of the different necessary software blocks will also be described in detail. This software uses third-party software, which was consciously chosen on the basis of its hardware regiments and type of license. This project was designed with the intention of being open source. The purpose of this block is to design embedded software that implements a high-performance motor controller and a sensor acquisition system, interfaced using high-speed buses and standard communication protocols. As a result, reliable, maintainable, efficient, and well-documented software is expected.

## 3.2 Requirement gathering

This project has started with the aim of covering some of the most important needs in control system laboratories. In the same way as in the hardware chapter, after many meetings with the parties involved, the following software requirements shown in Table 3.1 have been gathered.

Table 3.1: Software requirements.

| ID | Requirement |
|---|---|
| | **Controller** |
| **SW-1** | The software must implement the control loops |
| **SW-1.1** | The software must implemented a direct controller |
| **SW-1.2** | The software must implemented a feedback loop |
| **SW-1.3** | The software must implemented a forward loop |
| **SW-1.4** | The software must implemented a parallel loop |
| **SW-2** | The system must control in position, speed and current |
| **SW-3** | The controller reference can be set externally or internal,<br>The internal reference is generated by an internal function generator |
| **SW-3.1** | The functions could be: delta, step, ramp, sine, cosine,<br>parabola, exponential and trapezoid |
| | **Communications** |
| **SW-4** | The software must implement method to configure the motors,<br>control loops, sensors and function generation |
| **SW-5** | The software must implement method to extract data<br>(controller output and sensor data) |
| **SW-6** | The communications can be carried out trough Ethernet<br>UDP (main), UART and CAN |
| **SW-7** | The system must use ROS2 (Robotic Operative System)<br>communications. |

## 3.3 Modeling

The first step in designing an embedded system is to define the underlying infrastructure and behavior. Modeling is the process of obtaining deep knowledge about a system through imitation to be able to predict and control that system [26]. In summary, a model specifies what a system is supposed to do.

One widely extended method for system modeling is the use of finite-state machines (FSM). FSMs model the behavior of the system using states and transitions among

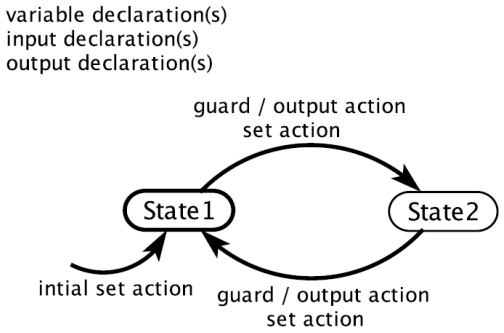



Figure 3.1: Typical finite state machine diagram [26].

those states. One of the main advantages of this modeling strategy is that this model can be translated directly into a mathematical model in which all possible states, transitions among them, and conditions can be checked using linear temporal logic (LTL). This is useful, especially when designing critical systems, because it allows us to predict the state of the system at every moment just by knowing the previous inputs. Therefore, the systems become fully predictable, allowing us to be sure that the system will behave as expected. Moreover, this modeling technique uses discrete states and discrete transitions that can be easily implemented in discrete devices such as processors and microcontrollers.

In a state machine, the state is at any moment the result of all previous inputs. All transitions among states are defined by a pair condition/action in which, when the condition is met, the system changes its state carrying out the action defined in the transition. The typical state diagram used to describe these models is shown in Figure 3.1. One of the main drawbacks of this model is that the number of required states increases with the complexity of the behavior, making its implementation difficult or even impossible in a real system. However, extended finite-state machines, also known as eFSM, solve this problem by adding the concept of internal variables, which can be read and updated in any transitions. As a result, the state of the system depends not only on the previous input, but also on the value of the internal variables.

### 3.3.1 System modeling

In this MSc Thesis, the system behavior was modeled using eFSM. The behavior of the whole system has been divided into two different state machines: communication and control. The communication machine is in charge of configuring the control machine and sending the output data, and the control machine implements the control loops. There are two identical control machines running at the same time, one per motor. The block diagram of these machines is shown in Figures 3.2 and 3.3.

### 3.3.2 Controller diagram

As requested in Table 3.1, the system implements four control loops: direct ($DIR$), feedback ($FB$), forward ($FF$) and parallel ($FP$). As a result, both motors are managed by the controller structure shown in Figure 3.4. The design of this module

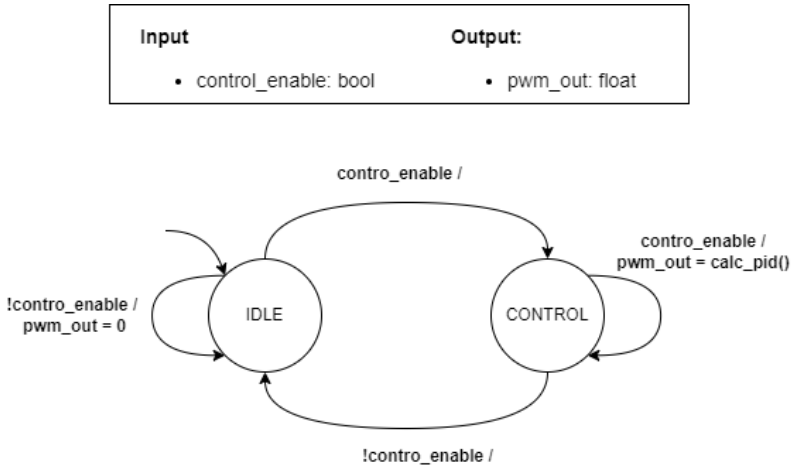



Figure 3.2: Finite state machine used in control.

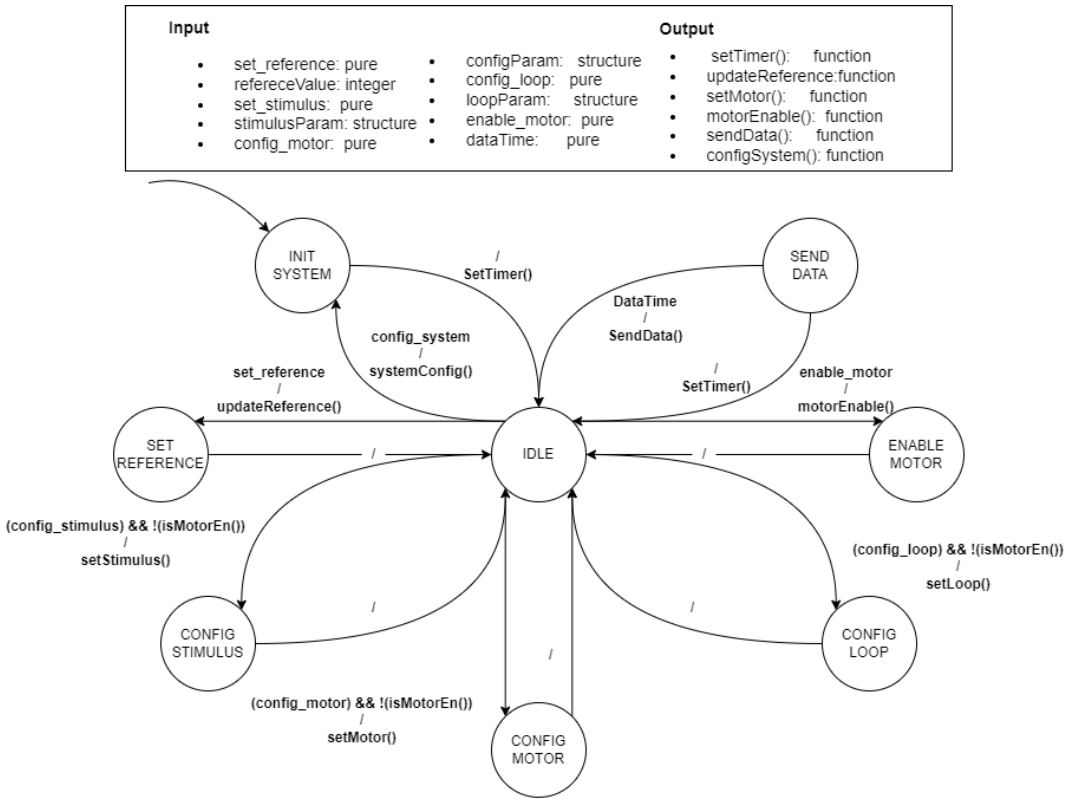



Figure 3.3: Finite state machine used in communications

has been based on the Telelabo implementation created by the RBZ embedded logics and Robolabo [18].

Both motor controllers are implemented in single-executed independent tasks that can be enabled and disabled according to the user's needs. Each loop implements a PID controller that can be configured independently. Every PID contains four configurable parameters: proportional constant ($K_p$), derivative constant ($K_d$),

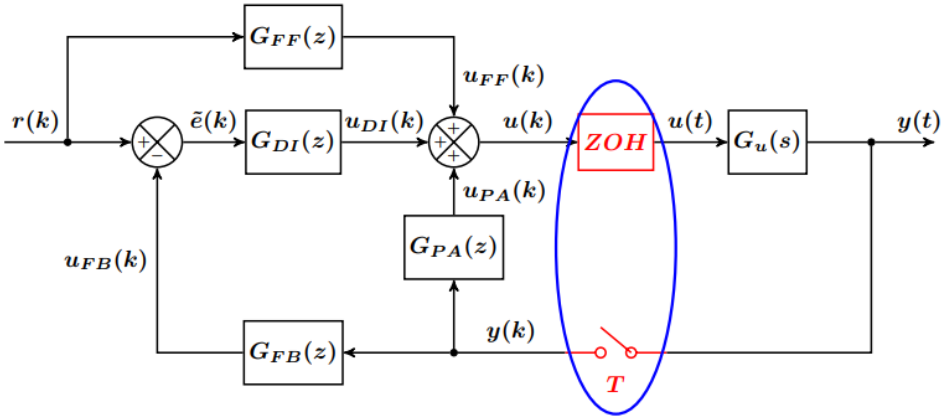



Figure 3.4: Control loop structure [18].

integral constant ($K_i$), and windup ($W_{up}$). The mathematical expression of a typical PID controller is shown in Expression 3.1. The windup variable is used to implement an anti-windup system on the integral controller. The integral windup occurs when the integral component accumulates a value beyond the saturation limits that causes the controller to stop working.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t) \tag{3.1}$$

The full controller allows three possible control schemes as follows:

- Position: the controller follows an angular position set by the reference ($r(k)$). This mode is used to move the motor to a very accurate position using the information provided by the encoder.

- Speed: the controller follows a speed set by the reference ($r(k)$). This mode is used mainly in applications in which a constant angular speed is required. The controller uses the information provided by the encoder for this purpose.

- Current: the controller will follow a specific current set by the reference ($r(k)$). This functionality is used in applications where torque control is required. This mode uses the information provided by the current feedback.

The controller also allows two possible reference sources: internal and external. When the internal reference is enabled, an internal stimulus generator is activated, generating a specific function on the reference input.This mode has been designed for the purpose of running tests and performing modeling tasks.

However, when the external reference is enabled, the user can override the reference value. This second mode has been implemented to perform real control applications in which the user can set a new goal according to its needs.

#### 3.3.2.1 Reference generator

As mentioned in the previous section, the controller incorporates an internal stimulus or function generator that can apply specific functions as a reference input. This mode facilitates testing and system modeling since external generators are no longer needed.

The reference generator can generate up to eight different types of function. The timestamp used to calculate the values is provided by the tick counter set by the controller. All functions have different parameters that can be modified. Information on the parameters required by the functions is shown in Table 3.2.

Table 3.2: Function list.

| Index | Name | Function |
|---|---|---|
| 0 | Delta | $y(t) = \begin{cases} 1 & t = 0 \\ 0 & t \neq 0 \end{cases}$ |
| 1 | Step | $y(t) = 1$ |
| 2 | Ramp | $y(t) = A \times t$ |
| 3 | Parabolic | $y(t) = A \times t^2$ |
| 4 | Sine | $y(t) = A \times sin(\omega \times t)$ |
| 5 | Cosine | $\mathrm{y(t)} = y(t) = A \times cos(\omega \times t)$ |
| 6 | Trapezoidal | $y(t) = \begin{cases} \frac{A}{T_1} \times t & t \leq T_1 \\ A & T_1 < t \leq T_2 \\ \frac{A}{(T_3 - T_2) \times (T_3 - t)} & T_2 < t \leq T_3 \\ 0 & t > T_3 \end{cases}$ |
| 7 | Exponential | $y(t) = v_1 + v_2 \times e^{-v_3 \times t}$ |

## 3.4 Robot Operating System: ROS2

The Robot Operative System (ROS) is a set of open-source tools and libraries that are used to help developers build and reuse code between robotic applications [5]. It includes services for hardware abstraction, low-level device control, message-passing between processes, package management, among others [51]. The main characteristics are the following:

- Uses a peer-to-peer (P2P) communication system with a publisher/subscriber structure.
- ROS is distributed, the publishers and subscribers can be anywhere.
- ROS is designed to be lightweight, the framework consists of standalone libraries wrapped around a thin ROS interface layer.

- ROS is language-agnostic; it can be used in any programming language. There are libraries for C, C++, Python, and Java, among others.
- ROS is free and open source.

The key advantage of ROS is that it creates a hardware abstraction layer that facilitates the development of complex systems without a deep knowledge of the hardware; low layers are standardized, making it easy to reproduce the results.

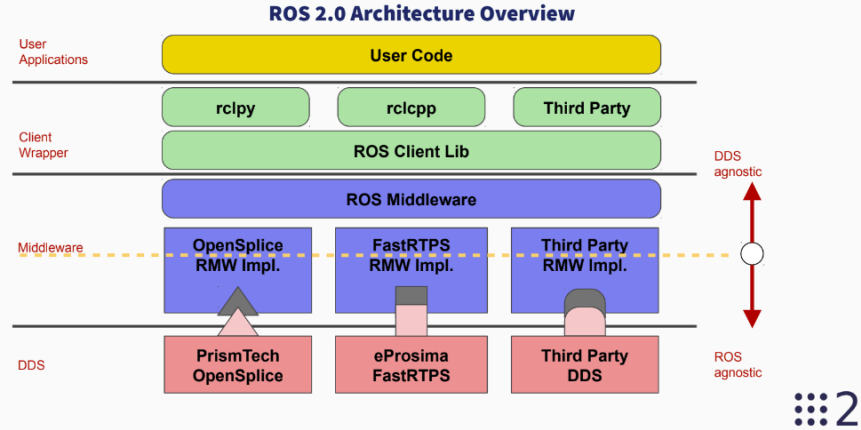



Figure 3.5: ROS2 architecture [2].

This open-source framework, started in 2007 at Stanford University, is supported by a global community of engineers, developers, and hobbyists who contribute to its development and maintenance. Therefore, ROS is becoming the standard in robotics programming. The blocks that shape the ROS framework can be classified into four main classes as follows [2]:

- Plumbing: this block is composed of all drivers, processes, message structures, and standards used not only to communicate the different software block, but also to interact with the hardware.
- Tools: it is composed of software for simulation, debugging, visualization, plotting, logging, and playback.
- Capabilities: this block consists of software use for control, planning, perception, manipulation, and mapping.
- Ecosystem: ROS includes a large global community that shares free software packages, tutorials, and knowledge. This is a powerful tool to speed up robot development.

### 3.4.1 ROS2 communications

ROS uses a publisher-subscriber structure composed of four key components [49]:

- Nodes: they are single-purpose executable programs that are individually compiled, executed, and managed.
- Topics: a topic can be defined as the name of a stream of messages, or in other words, they are the pipelines that the nodes use to communicate among them.
- Messages: data structures published on the topics. In the ROS framework, they are defined into .msg, .srv, and .action files.

There are two versions of the ROS framework; however, this project is based on the ROS2 framework. The ROS2 communication system is based on the DDS protocol. The Data Distribution System (DDS) is an API and middleware standard that provides connectivity, low latency, extreme reliability, and scalable architecture for Internet of Things (IoT) applications [16]. This communication middleware was introduced into the ROS2 framework, since DDS is an industrial standard that provides distributed system capabilities and security configurations, among others. As shown in Figure 3.6, ROS defines 4 possible ways to communicate between nodes:

- Publish-subscribe: nodes can publish on a topic and receive the data published by other nodes on the topics to which they are subscribed. This is a very scalable 1-N communications.
- Services: they provide call-and-response communications. Compared to the published/subscriber model, in which there is a continuous data stream, services only provide data when they are called by a client.
- Actions: they are intended to run a long task. Actions are similar to services, but they are preemptive and can provide continuous feedback while running.
- Parameters: parameters are configuration values of a node. A node can store integer, float, boolean, string, and list parameters. These parameters can be accessed and written by other nodes [50].

### 3.4.2 micro-ROS

ROS has been designed for use on systems with a minimum of hardware requirements like several hundred megabytes of RAM. In general, robots are not made up of one single, but numerous small microcontrollers that perform small tasks, such as managing access to sensors and actuators. These devices have only a few dozen kilobytes of RAM and limited processing capabilities, making it impossible to run ROS directly on them [25].

Micro-ROS is an open-source microcontroller-optimized framework that supports all major ROS capabilities and provides all libraries, tools, and interfaces required to integrate resource-constrained devices. Micro-ROS bridges the gap between resource-constrained systems and large processing units in robot applications [14].

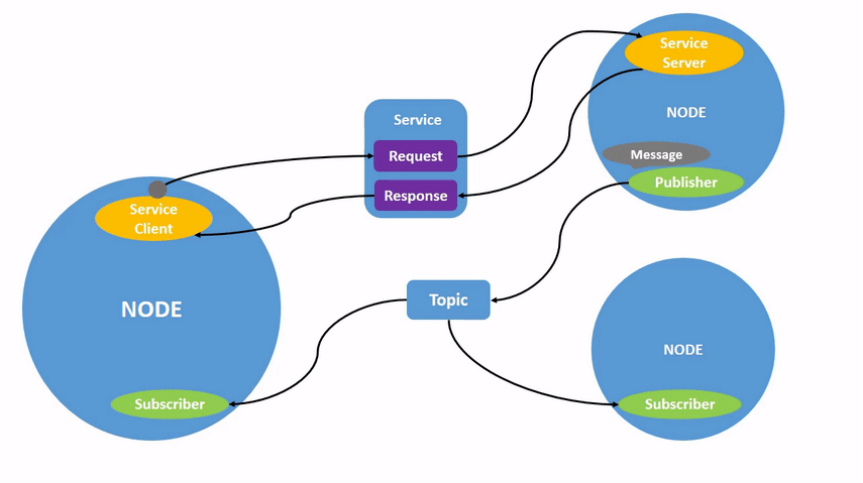



Figure 3.6: ROS2 communication diagram [50].

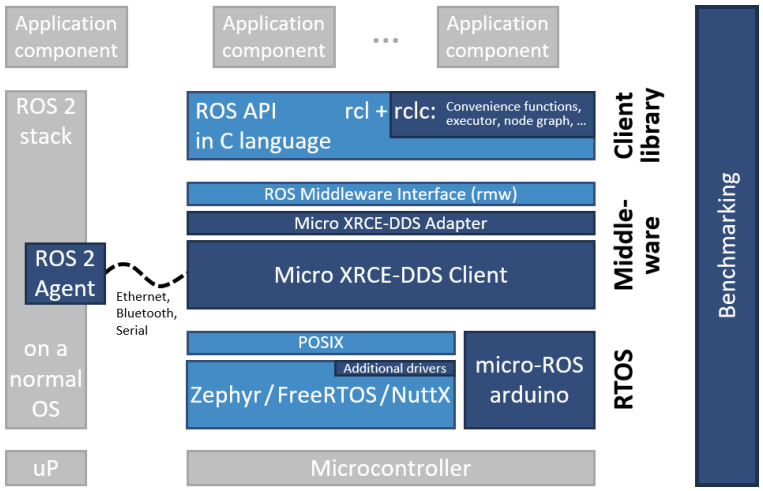



Figure 3.7: Micro-ROS architecture [14].

As shown in Figure 3.8, Micro-ROS follows the ROS2 architecture and makes use of its stack structure to use XRCE-DDS as communication protocol instead of DDS. XRCE-DDS stands for eXtremely Resource Constrained Environment Data Distribution System, and, as its name implies, it provides access to the DDS network from resource-constrained devices [32]. This is possible thanks to a client-server architecture, where low-resource devices (also known as XRCE Clients) are connected to a server (also known as XRCE Agent) which acts on behalf of its clients in the DDS network. In Figure 3.8, a general diagram of the XRCE-DDS structure is shown.

### 3.4.3 micro-ROS port

The micro-ROS middleware has been configured and compiled to run on the IMXRT1052. This has been done by creating a generic static library (using the micro-

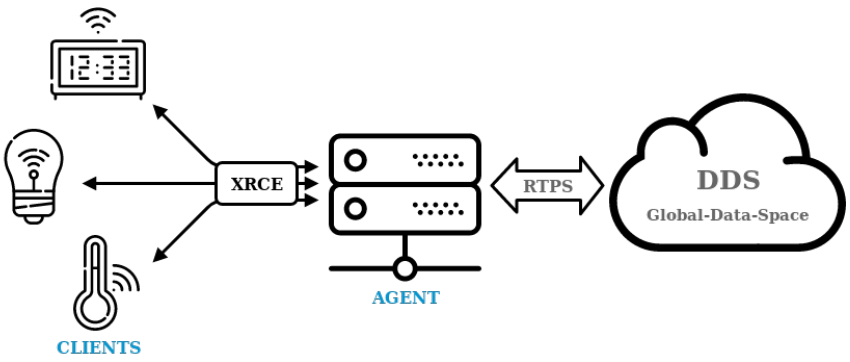



Figure 3.8: XRCE-DDS Structure [31].

ROS tools provided), editing the toolchain parameters, and providing the required dependencies. Micro-ROS provides 3 different transport options: UDP, TCP, and custom. The TCP, UDP transports require a posix-like socket library implementation. This socket library is provided by LwIP middleware, which has been included in the micro-ROS compile options. Customs transports can use any kind of bus as communication layer, it should just be implemented the open, read, write, and close functions according to the prototype functions provided by the micro-ROS library. In this project, not only are UDP transports used, but UART and CAN bus transports have also been implemented. The transport layer is selected at compilation time by editing the configuration macros.

micro-ROS is a static library, that is, all the required memory is allocated at compilation time. However, the publisher, subscribers, and services are allocated on run-time, which requires dynamic allocation. Hence, micro-ROS requires functions to allocate and free memory in a dynamic way. Given that micro-ROS runs on FreeRTOS, some native functions such as pvPortMalloc and pvPortFree have been used, and other custom functions have been implemented to provide memory reallocation capabilities. Lastly, required posix-like clock functions have been implemented using the FreeRTOS time counters.

All download, generation, and compilation processes were automated by a script, which can be executed in Linux-like operating systems and in Windows systems using the Windows Subsystem for Linux (WSL).

### 3.4.4 ROS application structure

The implemented ROS application consists of six services that are used to configure the system parameters, one service that extracts sensor data, and one publisher that extracts continuous data from the controller. Every service is identified by a topic that serves as a data stream for a specific message type. This application uses custom message types to configure and extract system data. In Table 3.3 there is a brief summary of the topics and data types used in this project. Furthermore, in Tables D.1, D.2, D.3, D.4, D.5 and D.6 there are descriptions of the message structure, data

types and expected values. Figure 3.9 shows a simple description of the implemented ROS2 application.

Table 3.3: Communication topics.

| | Topic | Data type | Description |
|---|---|---|---|
| | /motor/config | motor_data_msg/srv/MotorConfig | Set motor configuration |
| | /motor/sendData | motor_data_msg/srv/Enable | Enable continuous data sending |
| | /motor/enable | motor_data_msg/srv/Enable | Enable motor controller |
| Services | /motor/loop | motor_data_msg/srv/LoopConfig | Set loop configuration |
| | /motor/stimulus | motor_data_msg/srv/StimulusConfig | Set stimulus configuration |
| | /motor/sensoRead | motor_data_msg/srv/SensorData | Get sennsor data |
| | /motor/reference | motor_data_msg/srv/reference | Set controller goal |
| Topic | /motor/data | motor_data_msg/msg/MotorData | Controller data |

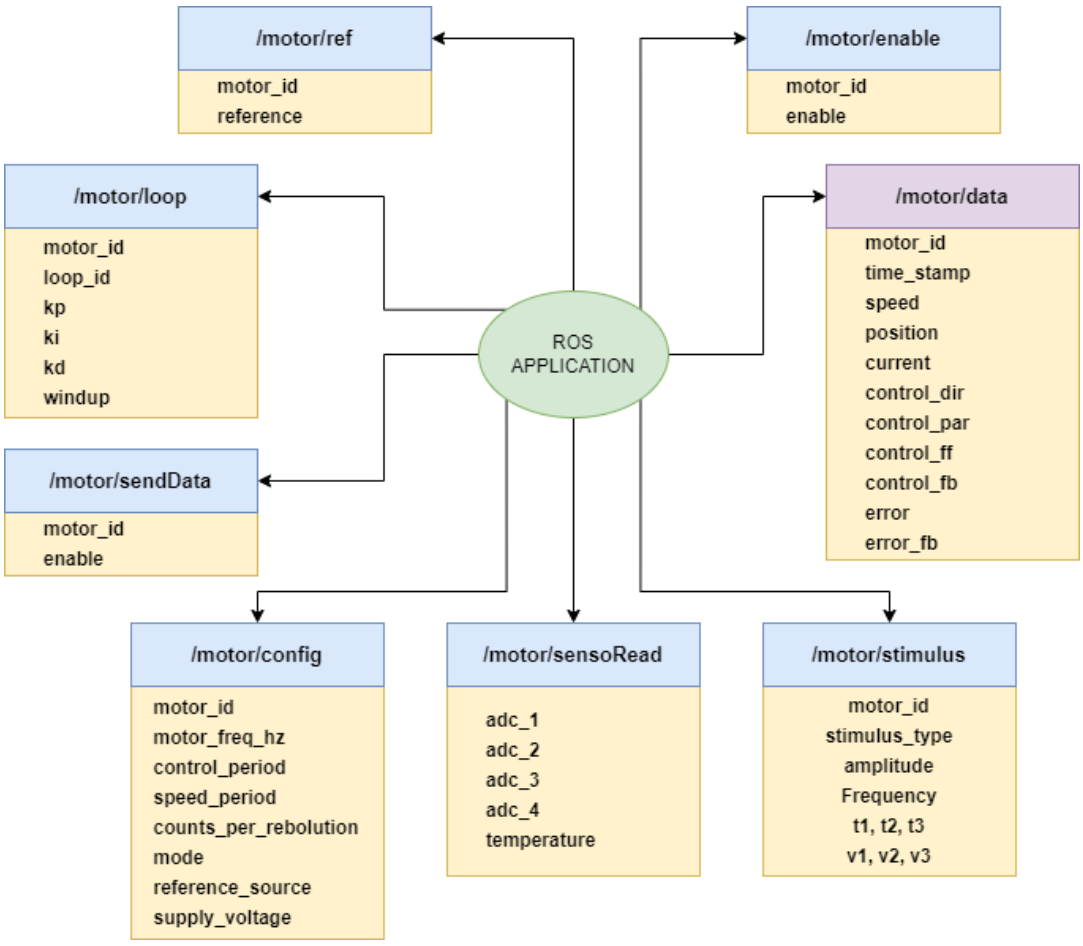



Figure 3.9: ROS application structure.

## 3.5 Drivers and middlewares

This software project has been created as the union of different software blocks. These software blocks can be classified into three different categories depending on their functions and the resources used. The three possible groups are as follows:

- Application: run the main tasks and the system logic. This software runs in a high-abstraction layer without direct interaction with the hardware layers.
- Middleware: this software adds functionalities, but it does not run at a high level as the application software or at a lower level as the driver software. The name middleware comes from the fact that it is in the middle of the application and the driver layer.
- Driver: this software interacts directly with the hardware, providing an abstraction layer for interacting with external devices and peripherals. Drivers implement the lower layer of the system.

In the following sections, the different middlewares and drivers (project-specific and third-party) will be described. Furthermore, the reasons why these software blocks have been selected and how they are integrated into the project will be explained.

### 3.5.1 Third party middlewares

#### 3.5.1.1 FreeRTOS

A real-time system is a system that can respond to an event in finite and bounded time. These events trigger a response that must be handled in a limit time, also known as deadline. Based on this definition, a real-time operating system (also known as RTOS) can be defined as an operating system that can meet these restrictive deadlines [13]. The main task of an RTOS is to schedule and execute tasks in a way that guarantees its deadlines.

FreeRTOS is an open source real-time operating system licensed under the MIT license. This lightweight implementation of an RTO is designed to be small, simple, and easy to use, making it suitable for running on embedded systems with low hardware resources. The size of a compiled binary of the FreeRTOS kernel is in the range of 4 KB to 9 KB. This RTO has been ported to the vast majority of hardware platforms from virtually all microcontroller manufacturers, making it a de facto standard in the industry [15].

FreeRTOS incorporates a real-timer preemptive task scheduler which allows to run multiple prioritized tasks concurrently in single core microcontrollers. Furthermore, it also provides tools for communicating (queues), synchronizing (events), and temporizing tasks, as well as tools for controlling access to shared resources (mutex and semaphores). The main advantages of using FreeRTOS are the following:

- FreeRTOS kernel is platform independent; that is, it can be compiled and executed on any hardware platform with enough hardware requirements.

- FreeRTOS is small and efficient, so it does not require high memory or processing capabilities.
- It is open source and free to use, even in commercial applications.
- It is a well-documented project with a huge community.
- its wide use shows its stability and reliability.
- Software implemented using FreeRTOS is easy to use, portable, and easily scalable.

This middleware is widely used in this project. On the one hand, all state machines and other auxiliary functions are separated into different tasks to guarantee their execution times. Furthermore, some of these tasks are communicated and synchronized using queue and events. One of the most used tools is the mutex; it is used to protect access to shared resources such as communication buses. Moreover, FreeRTOS is a requirement for using other middleware such as micro-ROS and LwIP.

#### 3.5.1.2 LwIP

LwIP is a small Low-weight independent and open-source implementation of the TCP/IP stack licensed under a BSD-Style license. Originally written by Adam Dunkels at the Swedish Institute of Computer Science, this is now developed and maintained by a huge developer community distributed all over the world.

LwIP has become a perfect solution for embedded devices, focusing on reducing memory usage. This full-scale TCP/IP stack implementation is suitable for use in systems with tens of kilobytes of free RAM and ROM for around 40 kilobytes of ROM. Its main features include:

- It includes support to protocols such as IPv4, IPv6, ICMP, TCP, UDP, IGMP. ARP, PPPosS, and PPPoE, among others.
- Includes support for the DHCP and DNS clients.
- Includes a Berkely-alike socket API, which is the de facto standard used in the POSIX sockets specification.
- Add-ons for applications such as HTTPS server, SNTP client, SMTP client, MQTT client, and TFTP server, among others.

Due to its multiple features and low weight, LwIP has been ported to a number of hardware platforms and operating systems; it can also run without an underlying operating system. The current versions of LwIP are fully compatible with FreeRTOS, which is the real-time operating system used on this project. LwIP is an essential part of this project, as it provides the networking capabilities used by other software blocks, such as micro-ROS. This project uses the TCP/UDP socket API and DHCP client features.

NXP SDK already includes the Ethernet interface layer required to port LwIP to the IMXRT1052 microcontroller. However, it was necessary to implement the Ethernet PHY driver, since the PHY chip used on the NXP development board was different from the one used in this project.

### 3.5.2 Project specific middlewares

#### 3.5.2.1 Pulse width modulator PWM

The NXP software development kit provides the necessary drivers to control their PWM peripherals; however, they are general purpose drivers that require quite a complex configuration. To enable a single channel of the PWM peripheral, it is necessary to configure the clock and the alternate function of the output pins, to connect those pins to the peripheral through the pin matrix, etc. This middleware has been created with the aim of abstracting that configuration process and providing a simple interface to control the peripheral. As a result, all configuration and control have been reduced to the functions shown in Listing 3.1. All functions listed use a PWM index parameter that identifies which PWM interface is being configured (PWM1 or PWM2). The resulting middleware headers are shown in Listing 3.1.

```
1  /** Initialize PWM peripheral with the given frequency. */
2  void exPWM_Init( PWM_Index_t pwmIndex, uint32_t frequency );
3
4  /** Set a duty cycle in a single channel  */
5  void exPWM_SetDutyCycle( PWM_Index_t pwmIndex, PWM_Channel_t
       channel, float  duty );
6
7  /** Set the duty cycle in both channel. */
8  void exPWM_SetDoubleDutyCycle( PWM_Index_t pwmIndex, float channelA
       , float channelB );
9
10 /** Set the PWM peripheral frequency. */
11 void exPWM_SetFrequency( PWM_Index_t pwmIndex, uint32_t frequency )
       ;
```

Listing 3.1: PWM middleware headers.

The source code is implemented in the pwm.c and pwm.h files allocated in the source folder. This middleware uses the PWM driver provided by the NXP SDK.

#### 3.5.2.2 Encoder controller

The IMXRT microcontroller used in this project contains peripherals specially designed to count incoming pulses from the rotatory encoders. The count value is stored into an internal 32 bit register. When the rotary encoder moves forward, the register value increases; on the contrary, when the encoder moves backward, the value decreases. The goal of this middleware is to convert those pulses into a relative angular position, angular speed, and direction.

This peripheral has two possible interrupt sources that can be activated when the counter register is underflowing (rollunder) and when the counter reaches a specific comparison value (rollover), which can be set via software. In the running state, both interrupts are enabled, and the comparison value is set to the number of counts per revolution. When the rollover interrupt is triggered, an internal variable is incremented; in the same way, when the rollunder triggers, the variable is decremented. As it can be seen, that variable is counting the number of revolutions. The value of this variable, together with the value of the counter register, provides the relative position of the encoder at any time. This is an efficient way to measure the relative position, avoiding potential problems such as overflow.

The speed is measured using the position and time difference between two consecutive measurements. This measure must be performed at regular time intervals to achieve a reliable measurement. Moreover, the time interval determines the resolution of the measurement. The longer the time, the greater the position difference. The speed measurement has been implemented using a periodic timer interrupt whose period can be defined in run-time.

The implemented middleware can control both encoder inputs independently at the same time. The peripherals are automatically configured by just providing the encoder index (ENCODER_1 or ENCODER_2), the count per revolution value (CPR), and the period used to measure the speed. The middleware provides functions to read the position in radians and the speed in radians per second, as well as many other auxiliary functions. The source code for this middleware is found in the encoder.c and encoder.h files located in the source folder. The resulting API headers are shown in Listing 3.2.

```
1
2  /** Get the pulse register value (ticks) */
3  uint32_t ENCODER_ReadCounter( ENCODER_Index_t encoder );
4
5  /** Get the internal pulse counter */
6  int64_t ENCODER_GetPosition( ENCODER_Index_t encoderIndex );
7
8  /** Get the relative position in radians */
9  float ENCODER_GetAngularPosition( ENCODER_Index_t encoderIndex );
10
11 /** Get the motor speed in radians per second */
12 float ENCODER_GetAngularSpeed( ENCODER_Index_t encoderIndex );
13
14 /** Get the number of pulses between two speed measurements */
15 uint32_t ENCODER_GetDifferentialPosition( ENCODER_Index_t
       encoderIndex );
16
17 /** Get the movement directions, '0' clockwise, '1' anticlockwise
       */
18 ENCODER_Direction_t ENCODER_GetDirection(ENCODER_Index_t
       encoderIndex);
```

```
19
20 /** Initialize encoder peripheral */
21 void ENCOER_Init( ENCODER_Index_t encoderIndex,
22                   uint16_t countPerRebolution,
23                   uint16_t speedSamplePeriod_ms );
```

Listing 3.2: Encoder middleware headers.

#### 3.5.2.3 Board general controller

This middleware has been designed to control general-purpose inputs / outputs (GPIOs) used on the boards. Hence, this software block provides functions to control the board LEDS, the analog multiplexer output, the fan, and the general-purpose output attached to the main connector.

#### 3.5.2.4 Timer service

As shown in previous sections, some software blocks require timer interrupts to handle some tasks. However, the number of hardware timers is limited and the configuration may be complex. This middleware has been created with the aim of providing a simple way to create hardware timer alarms, abstracting the configuration process and interrupt handling. Moreover, this software offers a common interface for initializing alarms, abstracting the hardware layer.

This software uses the programmable time interval peripheral (PIT), available on the IMXRT1052 microcontroller. This peripheral is based on a simple counter, which triggers an interrupt when a certain value is reached.

The middleware is capable of managing up to four alarms that can be enabled, disabled, or deleted. The Initialization of an alarm only requires the trigger period, the callback function, and a pointer to the argument passed to the callback function. The period resolution has been set to 1 ms due to the fact that a lower time resolution may cause processing overhead, which may affect the timing of the other task running. By default, alarms are configured to be triggered periodically during the time they are enabled. The resulting API headers are shown in Listing 3.3.

```
1 /** Initialize timer service. */
2 void TIME_SERVICE_Init( void );
3
4 /** Set a new timer. returns the the timer index or -1 if error. */
5 int TIME_SERVICE_SetTimer(TIME_SERVICE_Callback_t callback, void*
      param, uint16_t period );
6
7 /** Start the timer with the given index. */
8 void TIME_SERVICE_StartTimer( int timerIndex );
9
10 /** Stop the timer with the given index. */
11 void TIME_SERVICE_StopTimer( int timerIndex );
```

```
12
13 /** Delete the timer with the given index. */
14 void TIME_SERVICE_DeleteTimer( int timerIndex );
```

Listing 3.3: Timer middleware headers.

### 3.5.3 Device drivers

Drivers are software that is responsible for setting up and managing hardware devices at the lowest level. In this project, two different types of drivers are used. On the one hand, peripheral drivers provided by the microcontroller manufacturer are used to interface hardware devices attached to the communication buses.

On the other hand, drivers have been designed and implemented to manage the specific hardware devices used on the board. All of these drivers have been designed following a similar design pattern in which the input/output functions and the project/platform dependencies are extracted from the main source. Isolating the platform-dependent functions makes it easy for the software to be ported to other programs regardless of its architecture. In addition, these drivers use fixed-size variables (such as int16, int32, etc.) instead of basic datatypes (such as int, char, etc.) to avoid potential porting problems.

#### 3.5.3.1 Analog to digital converter MCP3564

This analog-to-digital converter manufactured by Microchip uses an SPI register-based structure. This device has a total of 16 internal registers made of volatile memory. These registers have sizes of 8, 24, and 32 bits. This driver has been designed to work with the entire MCP3561/2/4 ADC family [37].

All communication starts with an 8-bit command on the SDI input. This command defines the action that the interface will execute. The first two bits [7:6] are the separate addresses to which the device can respond. The following 4 bits [5:2] can serve two purposes: in the case of register read/write, they define the first address that is read/written; in the case of a fast command, they define what command is executed. Lastly, the remaining two bits determine the type of command that can be read static, read incremental, write incremental, and fast commands.

All registers are configured according to the specification given in the datasheet using bit masks. The different masks have been grouped into enumeration types to facilitate the configuration process and avoid confusion. The resulting driver provides functions to read voltage values and configure measurement type, gain, and oversampling, among many other functionalities. The porting layer implementation uses the SPI peripheral driver provided by the NXP SDK and the SPI3 peripheral.

The source code is allocated in the MCP356x.c and MCP356x.h files, and the porting functions are allocated in the MCP356x_port.c and MCP356x_port.h files. The resulting driver headers are shown in Listing 3.4.

```
/** Initialize device with default configuration. */
MCP356X_Error_t MCP356x_InitDefault( void );

/** Configure single channel measurement*/
MCP356X_Error_t MCP356X_SetSinegleChannel( MCP356X_Input_Source_t
    channel );

/** Read raw voltage in one shot mode*/
MCP356X_Error_t MCP356x_ReadSingle( int32_t* rawVoltage );

/** Read voltage */
MCP356X_Error_t MCP356x_ReadVoltage( float* voltage );

/** Set ADC operation mode */
MCP356X_Error_t MCP356X_SetWorkingMode( MCP356X_MODE_t mode );

/** Set convertion mode */
MCP356X_Error_t MCP3565X_SetConversionMode( MCP356X_CONV_MODE_t
    conversionMode );

/** Wait until measurement is completed */
MCP356X_Error_t MCP3565X_WaitForSample( uint32_t timeout );

/** Set oversampling rate */
MCP356X_Error_t MCP356x_SetOSR( MCP356X_OVERSAMPLE_t osr );

/** Configure differential channel measurement */
MCP356X_Error_t MCP356X_SetDifferentialChannel(
    MCP356X_Input_Source_t p_channel, MCP356X_Input_Source_t
    n_channel );

/** Configure interrupt pin */
MCP356X_Error_t MCP356X_SetIRQMode( MCP356X_IRQ_Mode_t mode );

/** Set MDAT pin output */
MCP356X_Error_t MCP356X_EnableMDAT( void );

/** Set measurement gain */
MCP356X_Error_t MCP356X_SetGain( MCP356X_GAIN_t gain );
```

Listing 3.4: MCP356X driver headers.

#### 3.5.3.2 H-Bridge HB2001

This H-Bridge monolithic device uses a fixed 16 bit SPI register-based communication structure. This device has a total of 4 volatile registers with a 13 bit size

[40]. Every transaction consists of a 16 bit data frame in which the first bit [15:15] determines the type of operation ('1' read, '0' write), the following two bits [14:13] determine the register to read/write and the remaining bits [12:0] are the register data. When a read operation is performed, the value of the register data bits is ignored.

This driver uses a bit mask-based structure in which the different masks are grouped to ease the use of the driver. The configuration of the registers follows the requirements specified in the device datasheet. The porting layer implementation uses the SPI peripheral driver provided by the NXP SDK and the SPI4 peripheral.

The source code of this driver is allocated in the MC33HB2001.c and MC33HB2001.h files, and the porting layer in the MC33HB2001_port.c and MC33HB2001_port.h files. The resulting driver headers are shown in Listing 3.5.

```
1  /* Initialize driver */
2  MC33HB2001_Error_t MC33HB2001_Init( void );
3
4  /** Get device ID */
5  MC33HB2001_Error_t MC33HB2001_DeviceID( uint16_t* devID );
6
7  /** Get system status */
8  MC33HB2001_Error_t MC33HB2001_GetStatus( uint16_t* status );
9
10 /** Check if the load is connected */
11 MC33HB2001_Error_t MC33HB2001_CheckOpenLoad( void );
12
13 /** Enable/Disable thermal managenet capabilities */
14 MC33HB2001_Error_t MC33HB2001_EnableThermalManagement( void );
15
16 /** Enable/Disable Current limit */
17 MC33HB2001_Error_t MC33HB2001_EnableCurrentLimit( void );
18
19 /** Set current limit */
20 MC33HB2001_Error_t MC33HB2001_SetCurrentLimit( MC33HB2001_ILIM_t
       iLimit );
21
22 /** Set slew rate */
23 MC33HB2001_Error_t MC33HB2001_SetSlewRate( MC33HB2001_SR_t sr );
24
25 /** Set operation mode */
26 MC33HB2001_Error_t MC33HB2001_SetMode(MC33HB2001_Mode_t mode );
27
28 /** Enable/disable output */
29 MC33HB2001_Error_t MC33HB2001_EnableOutput( void );
30
31 /** Enable/disable virtual input */
32 MC33HB2001_Error_t MC33HB2001_EnableVirturalInput( void );
33
```

```
/** Set virtual input value. */
MC33HB2001_Error_t
MC33HB2001_SetVirturalInput( MC33HB2001_VirtualInput_t vin, uint8_t
    value );
```

Listing 3.5: MC33HB2001 driver headers.

#### 3.5.3.3 Temperature sensor MCP9808

This device manufactured by Microchip is a digital temperature sensor that works with an I2C register-based structure [33]. This device has eight registers with a size of 16 bits that allow reading the device status, setting up the configurations, and reading the current temperature.

The standard I2C communications start with a 7 bit address followed by the operation bit ('0' read,'1' write). To write a register, the communications start with a write operation followed by the register address and the 16 bit register data. To read a register, the communications start with a write operation followed by the register address and a 16 bit read operation.

This driver uses a bit mask-based structure in which the different masks are grouped to facilitate the use of the driver. The resulting driver header is shown in Listing 3.6.

```
/** Initialize device with given address */
MCP9808_Error_t MCP9808_Init( uint8_t devAddress );

/** Configure device address */
MCP9808_Error_t MCP9808_setDevAddress( uint8_t devAddress );

/** Read temperature */
MCP9808_Error_t MCP9808_ReadTemperature( float* temperature );

/** Get/Set Critical temperature alarm */
MCP9808_Error_t MCP9808_GetCriticalTemperature( float* temperature
    );

/** Get/Set window temperature */
MCP9808_Error_t MCP9808_GetWindowTemperature( float* upperTemp,
                                              float* lowerTemp );

/** Get/Set hysteresis */
MCP9808_Error_t MCP9808_GetHysteresis( MCP9808_Hysteresis_t*
    hysteresis );

/** Get/Set resolution */
MCP9808_Error_t MCP9808_GetResolution( MCP9808_Resolution_t*
    resolution );

```

```
23 /** Get device identifier */
24 MCP9808_Error_t MCP9808_GetID( uint8_t* id, uint8_t* revision );
```

Listing 3.6: MCP9808 driver headers.

## 3.6 Project structure

The software folder structure is shown in Figure 3.10. This project structure has been generated by the automatic project creator included in the integrated development environment provided by NXP known as MCUXpresso. This set of tools is responsible for the import and configuration of peripheral drivers, third-party libraries, and middlewares such as LwIP and FreeRTOS.

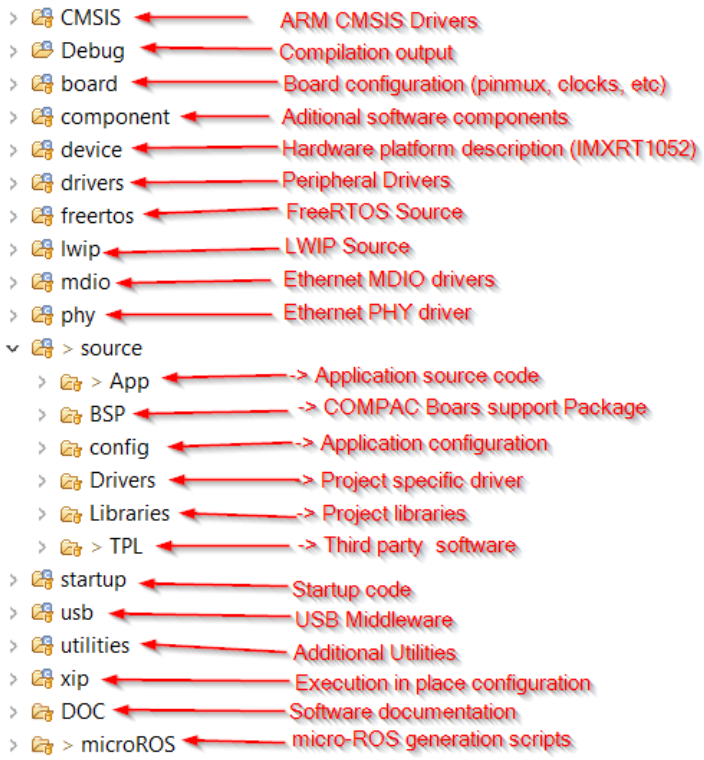



Figure 3.10: Project folder structure.

The vast majority of code specifically developed for this project is located in the source folder. The 'App' folder contains the application logic such as hardware initialization, communications configuration, control task, project-specific middleware, among others. The BSP folder contains the board support package provided by Simplia, with different useful drivers and middlewares used on this project. The Driver folder contains the drivers specifically developed for this project. The config folder contains all the necessary headers used to configure the project. The library folder contains additional libraries required by the board support package. Lastly, the TPL folder contains the third-party software used in this application.

# Chapter 4

# Testing

## 4.1 Introduction

During the development process, there may be some errors. These errors can occur in design, manufacturing, or assembly processes or even during the definition of the concept. The hard task is not how many errors have been made, but how to detect and solve them. Although some errors, such as assembly issues, can be quickly resolved, others, such as design errors, can be tedious to solve.

The way the system is tested is crucial to delimit the origin of failures and to avoid the creation of new problems. Some failures that can be easily solved, such as short circuits, can have disastrous consequences if they are not detected in time. Moreover, some problems that can be classified as a software bug at first glance may have its origin in a non-detected hardware issue. Detecting those problems in the correct step can save a lot of debugging time.

As can be seen, it is important not only the test that is performed, but also the order in which they are performed. This chapter will describe the testing process carried out in this MSc. Thesis; it will start with a non-electrical testing in an effort to find potential problems at a glance. The power rails will then be tested and finally the software and functional tests will be performed. This process is not linear; as will be seen, some problems require more than one iteration to figure out its origin.

## 4.2 Visual inspection

This is the first simple test to find obvious problems at first sight. It is important to perform this test before electrical testing to avoid electrical damage or malfunctions when connecting the power supply. The checklist passed on this test is as follows:

- Incorrectly placed components: all components must be soldered correctly without assembly problems and aligned well with the footprint pads. The goal is to find problems such as component misalignment and tombstoning.
- Missing components: all required components must be soldered. It will be checked that all components are in their correct place.

- Short circuit: there must be no solder bridges between two consecutive pins or neighbor pads. In almost all cases, these bridges can be easily seen.
- Board integrity: components and board traces cannot be destroyed or damaged. Excess heat during the assembly process can destroy components and circuits; some of these problems can be easily detected.

The boards did not show critical errors except for some missing resistors.

## 4.3 Electrical inspection

Once the visual inspection has passed, the next step is to perform electrical testing. These tests will look for electrical problems that are not evident at first sight. To avoid electrical damage, the first tests were carried out without connecting the main supply using a digital multimeter. The checklist passed on this test is as follows:

- Component polarity: all active components and integrated circuits should have the correct polarity and conduction value. It is necessary to check all protection, barrier, and light-emitting diodes.
- Short circuit: there must be no electrical connection between consecutive pins or pads, buses, or different power rails. It is especially important to check the short circuits between the power rails and the ground to avoid damage.

### 4.3.1 Power Supply

The power tree is one of the most important hardware blocks; any malfunction here can destroy or make the prototype unusable. Testing this block requires extreme care and the use of special equipment, such as a current-limited power supply, to avoid damage. It is worth mentioning that the reason why the +5V supply shows a voltage range lower than the nominal one is because of the voltage drop in the series diode used in the load switch.

Table 4.1: Voltage measured on different power rails.

| Rail | Minimum | Maximum | Average | Units |
|---|---|---|---|---|
| +3.3VA | 3.28 | 3.32 | 3.3 | V |
| +3.3V | 3.32 | 3.38 | 3.35 | V |
| +5V | 4.88 | 4.94 | 4.9 | V |

Table 4.1 shows the voltages measured on the different power rails, all voltages are in the operative range. As mentioned in previous chapters, one important parameter to take into consideration is the noise coupled in the power rails, especially in the analog power rail. Figures 4.1, 4.2, and 4.3 show the noise measured on the +3.3 V

analog, +3.3 V and +5 V power rails, respectively. The noise measured on the +3.3V analog supply has an average voltage of $224\mu V$ that, compared to the $1.47mV$ and $968\mu V$ measured on the +5V and +3.3V supplies, is much lower as required.

As a summary, it can be concluded that the power supplies work with the expected voltage ranges and noise.

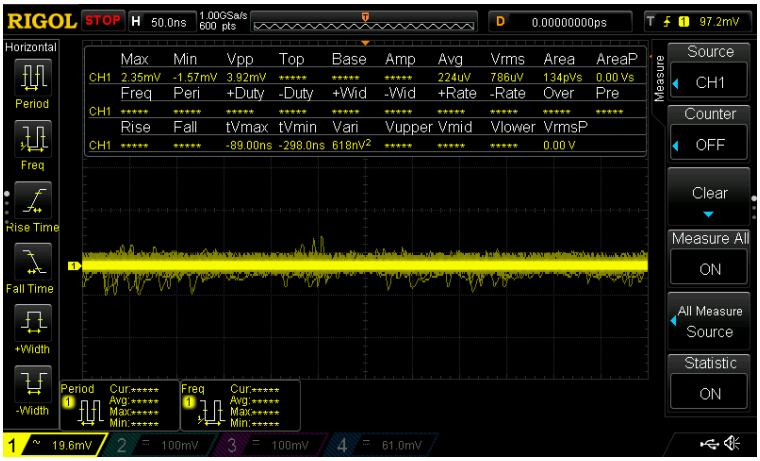



Figure 4.1: Noise measured on the analog +3.3V power rail.

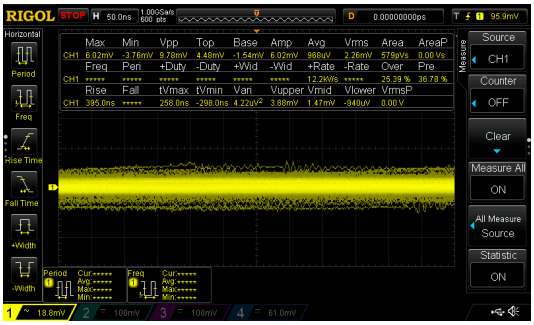



Figure 4.2: Noise measured on the +3.3V power rail.

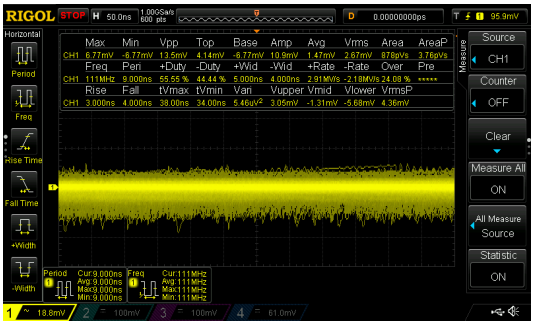



Figure 4.3: Noise measured on the +5V power rail.

## 4.4 Software testing

With the hardware fully tested, the next step is to check that all software blocks, peripherals, and devices work as expected. In some of these tests, more than one feature is checked at the same time because of the implicit relation among them for example, the testing of the drivers and middlewares carries with the testing of the communication buses.

The first test consists of flashing a simple code to verify the connection between the probe and the microcontroller and verifying the integrity of the internal memory. This test is important to check that there are no hardware issues and that the start-

up routines work as expected. As can be seen, this test is implicit in all other tests, since all of them require flashing the code; however, it is important to make a first connection test before starting to debug the software to discard potential failures.

Once the connection to the microcontroller is verified, the next step is to test the different parts of the software. The test performed and the procedures followed are summarized in the following sections. As a programmer and debugger, the J-LINK V9 USB debugger manufactured by Segger has been used.

### 4.4.1 SPI bus, ADC driver, and analog multiplexers

This test will verify that the SPI peripheral, the ADC driver, and the analog multiplexer work properly. When the analog multiplexer is enabled, the inputs of the even channels are connected to the resistor divider used to measure the strain gauges. The expected voltage is 1.65 V, since the power supply of the analog block is 3.3V and it is a balanced voltage divider. Before measuring channel 4, the gain is set to 1/3 to test this feature.

The code used to perform this test is shown in Listing 4.1. As shown in Figure 4.4, the measurements coincide with the expected values.

```
1  float ReadSingleChannel( MCP356X_Input_Source_t channel )
2  {
3      float voltage = 0;
4      /* Set single channel A configuration */
5      MCP356X_SetSinegleChannel(channel);
6      /* Start single measurement */
7      MCP3565X_SetConversionMode(MCP356X_CONV_MODE_ONESHOT3);
8      MCP356X_SetWorkingMode(MCP356X_MODE_CONVERSION);
9      /* Wait until measurement is completed */
10     MCP3565X_WaitForSample(ACQUISITION_TASK_READ_TIMEOUT);
11     /* Read and store data */
12     MCP356x_ReadVoltage(&voltage);
13
14     return voltage;
15 }
16
17 void ADCTest( void )
18 {
19     float measure1, measure2, measure3, measure4;
20
21     /* Init device */
22     MCP356x_InitDefault();
23
24     /* Enable analog muxes */
25     BOARD_CONTROL_EnableGauge(GAUGE_1);
26     BOARD_CONTROL_EnableGauge(GAUGE_2);
27     BOARD_CONTROL_EnableGauge(GAUGE_3);
28     BOARD_CONTROL_EnableGauge(GAUGE_4);
29
```

```
30      /* Read single channel */
31      measure1 = ReadSingleChannel(MCP356X_CHANNEL_CH0);
32      measure2 = ReadSingleChannel(MCP356X_CHANNEL_CH2);
33
34      /* Set ADC Gain to 1/3 */
35      MCP356X_SetGain(MCP356X_GAIN_0b3);
36      measure3 = ReadSingleChannel(MCP356X_CHANNEL_CH4);
37
38      /* Set ADC Gain to 1 */
39      MCP356X_SetGain(MCP356X_GAIN_X1);
40      measure4 = ReadSingleChannel(MCP356X_CHANNEL_CH6);
41  }
```

Listing 4.1: ADC test code.

| Name | Type | Value |
|---|---|---|
| measure1 | float | 1.66514373 |
| measure2 | float | 1.66621208 |
| measure3 | float | 0.559853554 |
| measure4 | float | 1.66972554 |

Figure 4.4: ADC test results.

### 4.4.2 PWM middleware

This test checks at the same time the PWM peripheral and the PWM middleware designed to manage it. As shown in Listing 4.2, the test consists of setting each PWM peripheral with a different frequency and then setting different duty cycles in each channel.

```
1  void testPWMGeneration( void )
2  {
3      exPWM_Init(PWM_INDEX_2, 30000UL);
4      exPWM_Init(PWM_INDEX_1, 40000UL);
5
6      exPWM_SetDoubleDutyCycle(PWM_INDEX_2, 20 , 40 );
7      exPWM_SetDoubleDutyCycle(PWM_INDEX_2, 60 , 80 );
8  }
```

Listing 4.2: PWM test code.

The results are checked using a digital analyzer attached to the PWM outputs. The generated PWM signals are shown in Figure 4.5 and, as can be seen, the modules work as expected.

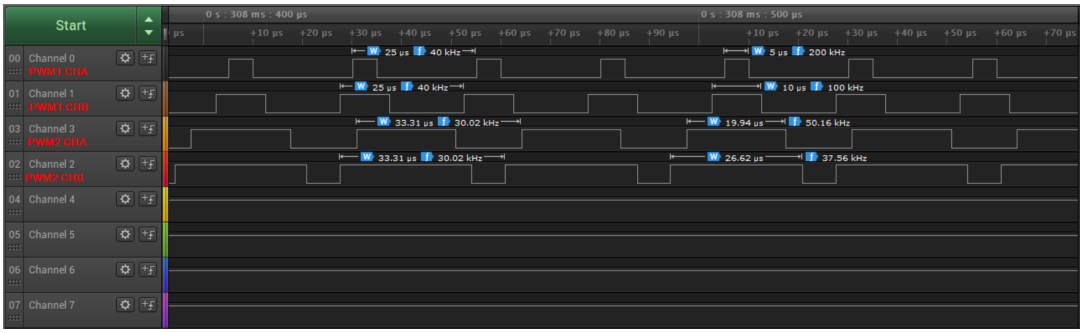



Figure 4.5: PWM test results.

### 4.4.3 UART peripheral

This test checks the peripheral UART functionality using an echo program: the program will reply the same data it receives until an escape character arrives. The test code used is shown in Listing 4.3.

```
1  void UARTTest( void )
2  {
3      char buffer = '\0';
4      uint16_t size = 0;
5      /* Initialize peripheral */
6      UART_Init(ACQUISITION_UART_INDEX,     /* UART1 */
7                ACQUISITION_UART_BAUDRATE,  /* 115200 */
8                ACQUISITION_UART_PARITY);   /* None */
9
10     while(buffer != 'q' )
11     {
12         size = UART_ReadData(ACQUISITION_UART_INDEX, &buffer, 1,
       10);
13         if( size != 0 )
14         {
15             UART_SendData(ACQUISITION_UART_INDEX, &buffer, 1, 10);
16         }
17     }
18 }
```

Listing 4.3: UART test code.

This test requires a UART-to-USB converter attached to the UART port and serial console software. The result of this test is shown in Figure 4.6. As can be seen, the peripheral seems to work correctly.

### 4.4.4 CAN communication

This test verifies the correct behavior of the CAN bus. It consist of an simple echo application that replies the received bytes plus one. Periodic messages were sent to the board using a CAN-to-USB converter and the CAN bus monitor. Figure 4.7

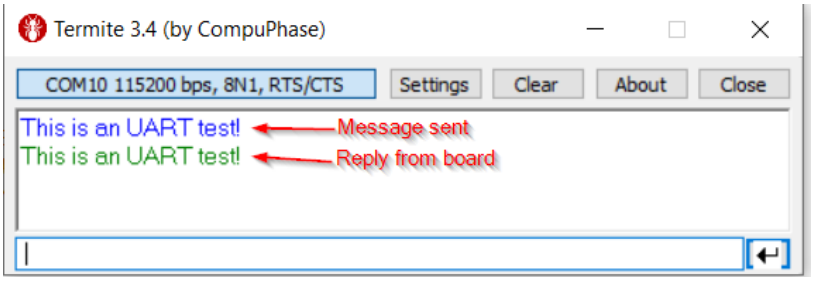



Figure 4.6: UART test results.

shows the messages sent and the replies from the board. It can be concluded that the bus works as expected working at a baudrate of 500000 bps.

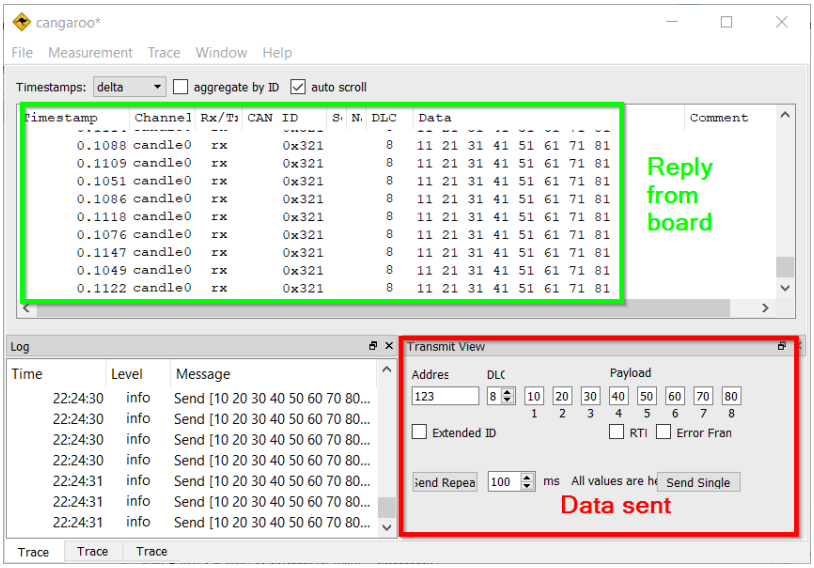



Figure 4.7: CAN bus test results.

### 4.4.5 I2C and MCP9808 driver

This test checks the functionality of the I2C peripheral and the MCP9808 temperature sensor driver. The test tries to set and read some device parameters and then try to get the current temperature. The code used to check these functionalities is presented in Listing 4.4.

```
1 void I2CTempTest( void )
2 {
3     uint8_t revision, id;
4
5     float wTempCritical = 35.75;
6     float rTempCritical = 0.0;
7     float temperature = 0.0;
8
```

```
9       MCP9808_Init(0x18);
10      /* Get device data */
11      MCP9808_GetID(&id, &revision);
12      /* Remove write protection */
13      MCP9808_UnlockCriticalTempReg();
14      /* Set and check critical temperature alarm */
15      MCP9808_SetCriticalTemperature(wTempCritical);
16      MCP9808_GetCriticalTemperature(&rTempCritical);
17      /* Read current temperature */
18      MCP9808_ReadTemperature(&temperature);
19  }
```

Listing 4.4: I2C and MCP9808 driver test code.

This test was carried out using the debugger and with the power board attached to the control board. After some measurement tests, it can be concluded that the driver and peripheral work correctly.

### 4.4.6 Encoder middleware

This test checks the functionality of the encoder by measuring the position and speed of a motor that runs at its maximum speed (100% duty cycle). The code used in this test is shown in Listing 4.5.

```
1   void EncoderTest( void )
2   {
3       int counter = 100;
4       float position = 0.0, speed = 0.0;
5       /* Init encoder peripheral */
6       ENCOER_Init(ENCODER1, 3700, 10);
7
8       while(counter--)
9       {
10          /* Read data */
11          position = ENCODER_GetAngularPosition(ENCODER1);
12          speed = ENCODER_GetAngularSpeed(ENCODER1);
13      }
14  }
```

Listing 4.5: Encoder test code.

The test was repeated with the motor moving in opposite directions to verify that the middleware works in both directions. The results shown in Figure 4.8 show that the peripheral and middleware work as expected.

### 4.4.7 USB bus

This test was designed to verify USB bus functionality. It creates a virtual serial port over USB and implements an echo application similar to the test used to test the UART bus. The code employed to test this device is shown in Listing 4.6.

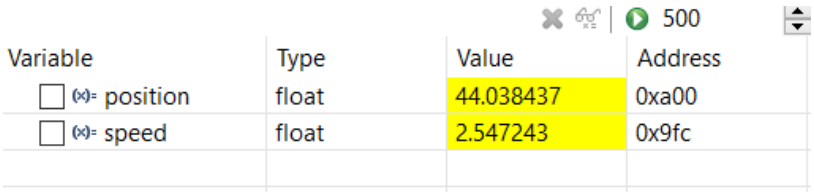



Figure 4.8: Encoder test results.

```
1  void USBVComTest( void )
2  {
3      VIRTUAL_VCOM_Init();
4      char buffer = '\0';
5      size_t size = 0;
6
7      while( buffer != 'q' )
8      {
9          size = VCOM_Read(&buffer, 1);
10         if(size != 0)
11         {
12             VCOM_Write(&buffer, 1);
13         }
14     }
15 }
```

Listing 4.6: USB test code.

This test requires the use of a serial communication console. The results of this test are shown in Figure 4.9. As can be seen, the test replies exactly the same string sent.

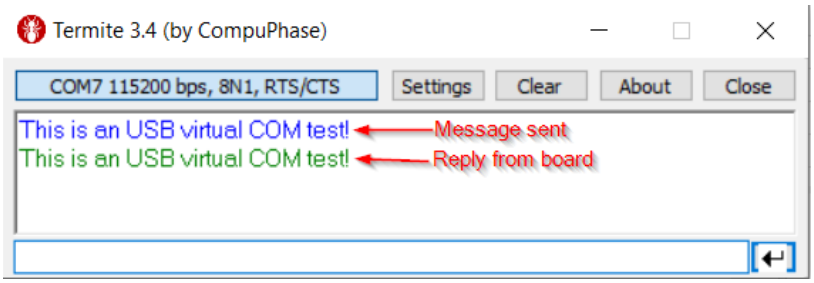



Figure 4.9: USB test results.

### 4.4.8 Ethernet and LwIP middleware

This test checks the functionalities of the ethernet peripheral and LwIP middleware. The software creates a socket on port 7 and starts listening for TCP traffic implemented an echo application: all incoming traffic is forwarded to the sender.

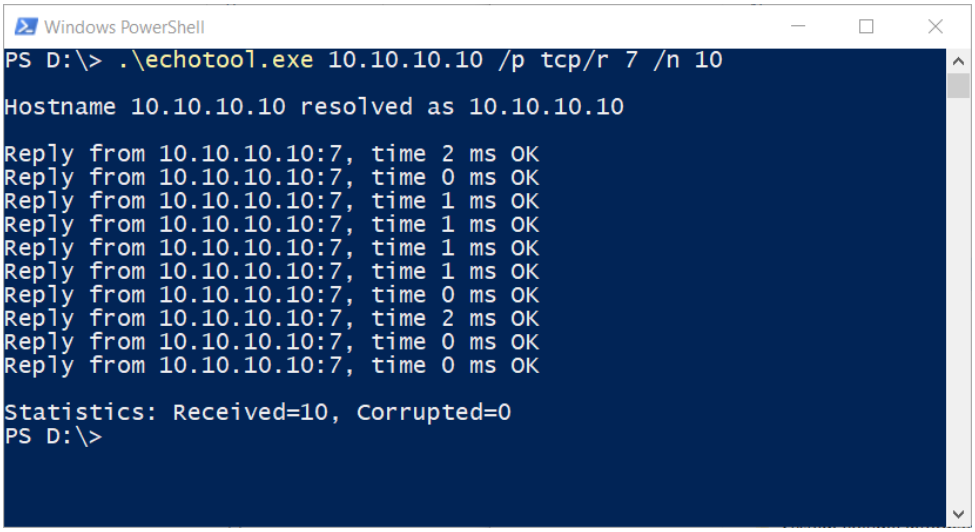



Figure 4.10: LwIP and ethernet test results.

The board has been connected through the Ethernet port to a computer, and both IPs have been configured in the same range. Then, using a TCP echo tool, the functionality was tested. In Figure 4.10 the test results can be seen.

### 4.4.9 ROS middleware

Having the hardware and communication buses tested, the next step is to verify the connection between micro-ROS middleware, the XRCE-DDS agent, and ROS2. To perform this test, it is necessary to have a full installation of ROS2, the agent software, and the message package designed for this application. This software and other required dependencies are automatically installed by the system setup script implemented for this project (see Appendix C).

First, the connection between the micro-ROS application and the XRCE-DDS agent was verified. The test was repeated to test the different communication buses used in this application (ethernet, UART, and CAN). The switch between these buses requires a recompilation of the micro-ROS middleware and the application. The result of the UART test is shown in Figure 4.11. This test was performed using a UART-to-USB adapter connected to the machine running the agent software, configured to work at a 576000 baudrate. As can be seen, the node, its services, and its publishers are registered correctly.

Finally, integration with the ROS2 environment was verified. This test consisted of trying to access the services, topics, and data types registered by the node. It was performed using the ROS2 command line application. In Figure 4.12, it can be seen that services and topics are registered as expected. In the last line, the message structure of the configuration service can be seen, which means that the application is also integrated with the message package.

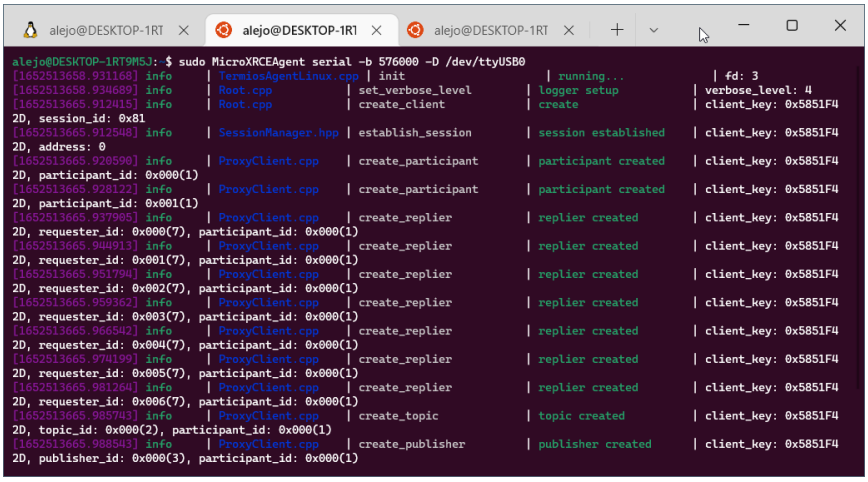



Figure 4.11: Agent connection test.

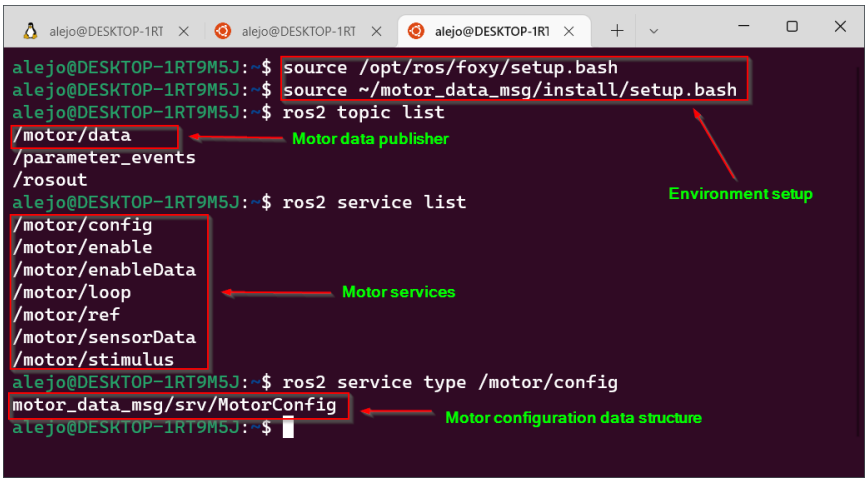



Figure 4.12: ROS2 connection test.

## 4.5 Functional test

The last step in testing is to verify that the application behaves as expected. For this purpose, a Python script has been developed to configure the motor parameters, subscribe to the data topic, and plot the captured data. This script calls to the node services using a Python ROS2 API included in the ROS2 middleware.

The first test was performed using only the control board without the power board. This test consisted of configuring the stimulus generator and plotting the generated function to verify the application behavior, the service configuration, and the communication bandwidth. In figures 4.13 and 4.14 the generation results for the trapezoidal and sine functions can be seen.

The last test consisted of measuring the motor response to different functions. This test required the connection of all hardware: control board, power board, motor, and

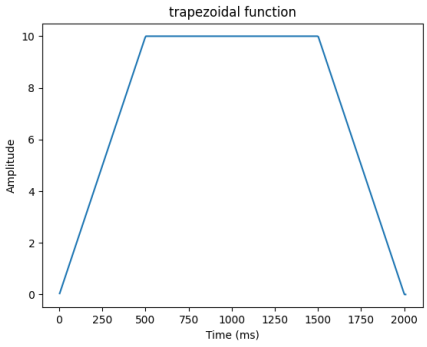



Figure 4.13: Trapezoidal function generation result.

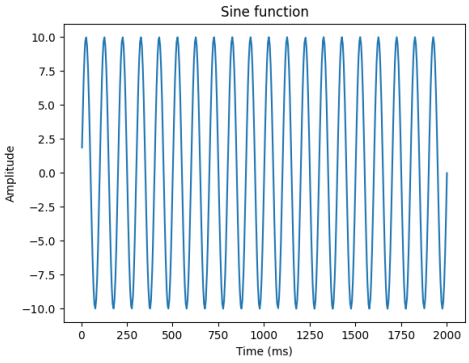



Figure 4.14: Sine function generation result.

encoder. The configuration used in the motor is shown in Table 4.2 and in the loop in Table 4.3. As can be seen, the system was configured with the simplest controller. All stimulus used were configured with an amplitude of 10 radians.

Table 4.2: Motor configuration used for testing.

| **Parameter** | **Value** | **Unit** |
|---|---|---|
| Motor ID | MOTOR1 | - |
| Frequency | 3000 | Hz |
| Control Period | 1 | ms |
| Speed Period | 30 | ms |
| CPR | 3700 | pulses |
| Mode | Position | - |
| Reference | Internal | - |
| Voltage | 12 | V |

Table 4.3: Controller loops configuration used for testing.

| **Loop** | **Kp** | **Ki** | **Kd** | **Windup** |
|---|---|---|---|---|
| Direct | 1 | 0 | 0 | 12 |
| Feedback | 1 | 0 | 0 | 12 |
| Feedfordward | 0 | 0 | 0 | 0 |
| Paraller | 0 | 0 | 0 | 0 |

In Figures 4.15, 4.16 and 4.17 the motor responses to the steps, ramps, and parabola functions are shown, respectively. The quantification noise shown in the speed measurements is due to the low pulse resolution of the encoder: between two consecutive measurements, there are not enough pulses to have a high resolution. The adjustment of the time between two consecutive speed measurements depends on the encoder CPR used, and in the number of points required, the higher time means the lesser measurements.

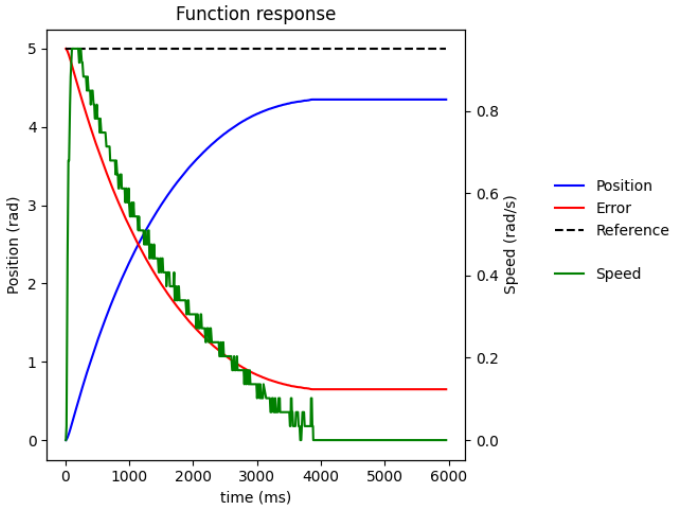



Figure 4.15: Step response.

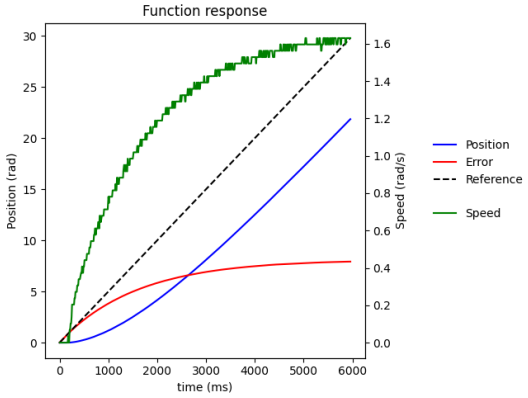



Figure 4.16: Ramp response.

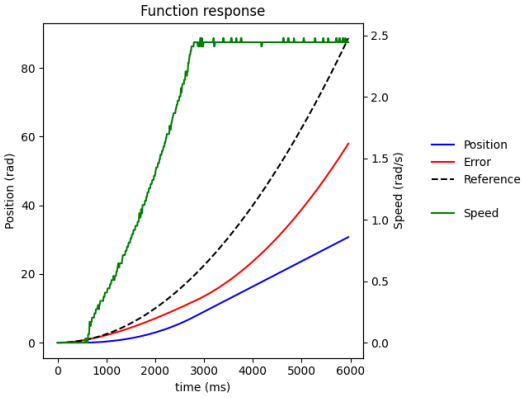



Figure 4.17: Parabola response.

## 4.6 List of errata

After the testing process, there were some problems that, due to their type, could not be solved. However, none of these errors was blocking, only some features were affected, allowing the development of the main concept. These problems are as follows:

- P-MOS transistor N2 is inverted. The drain and source pins are inverted, which causes the load switch to not work properly. This problem was found during power supply testing. Solving this problem requires a redesign of the board. Figure 4.18 shows the current schematic and Figure 4.19 shows the corrected schematic. Furthermore, Appendix H shows the complete board schematic.

- The COMPAC X14 connector is inverted. This connector is flipped 180 degrees, causing a loss of SPI and GPIO capabilities in the carrier board connector. This problem limits some features, such as the fan controller and the SPI functions of the HB2001 driver; however, they are not critical. This problem was found during software testing. Figure 4.20 shows the current position of the connector. It can be seen that the first pin is in the bottom-left corner. Instead, in the

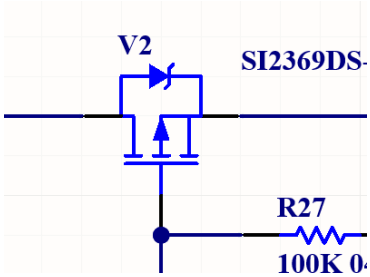



Figure 4.18: Transistor N2 flipped.

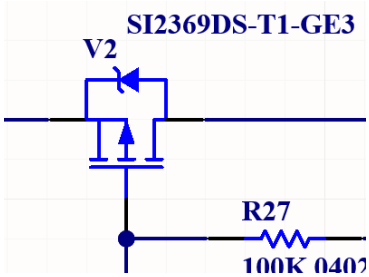



Figure 4.19: Transistor N2 corrected.

corrected layout shown in Figure 4.21, it can be seen that the first pin is located in the top-right corner.

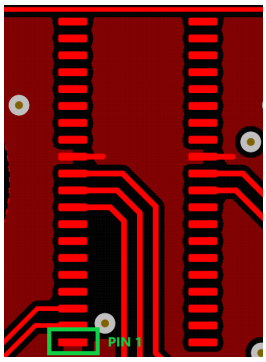



Figure 4.20: Connector X14 flipped.

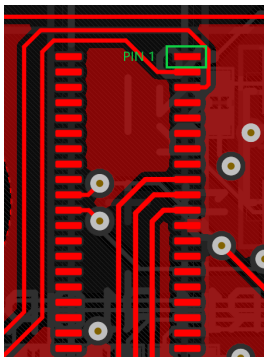



Figure 4.21: Connector X14 corrected.

- Operational amplifiers N1 and N3 have incorrect reference. The reference of the mounted component is MCP6401T instead of MCP6401RT, causing a polarity inversion of the power supply. This problem was found during power testing and could not be solved due to this component, and compatible ones are out of stock. Figure 4.22 shows a comparison between both components where it can be seen that they have the power polarity inverted.

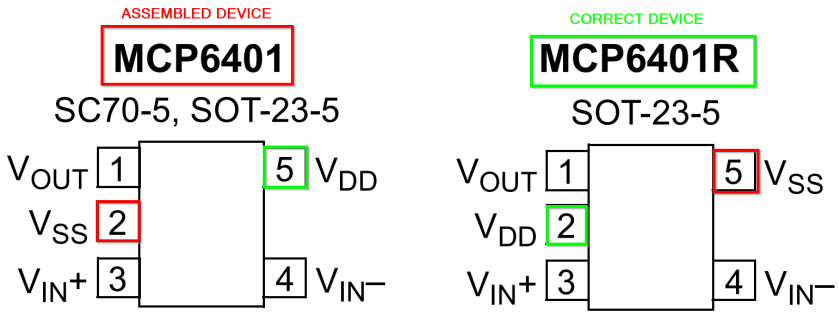



Figure 4.22: Comparison between MCP6401 and MCP6401R.

- The MIC4606 is connected to a wrong power rail. Although the datasheet shows that this device can withstand voltages in the range of -0.3 to 18 volts, it requires an operating voltage of 5.5 to 16 volts to work properly [36]. In this project, this device has been connected to the 3.3 V power rail, out of the operating range. Figure 4.23 shows the current schematic, Figure 4.24 shows the schematic with the necessary corrections, and Appendix H shows the full board schematic. This problem cannot be fixed without a board redesign, so this board could not be used and tested in this MCs. thesis. Instead, the HB2001 board was used to perform functional tests.

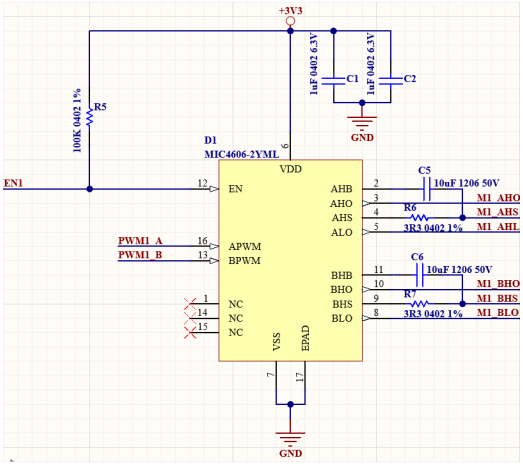



Figure 4.23: Wrong MIC4606 schematic.

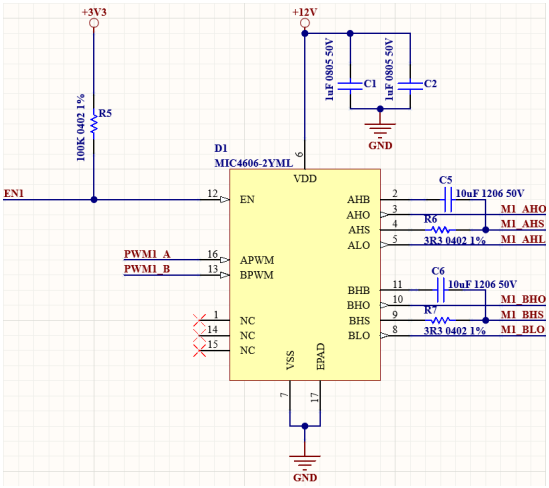



Figure 4.24: Corrected MIC4606 schematic.

# Chapter 5

# Conclusions

## 5.1 Conclusions

This project was developed with the aim of creating a control platform to facilitate and speed up the modeling, design, and implementation of brushed motor control systems. As shown in previous sections, there is a wide variety of parameters and design variables to take into account when designing this kind of platform.

This project involves all the necessary processes and tasks required in a real engineering project, from gathering user requirements to manufacturing and testing. It is worth highlighting that this project has been designed with a user-friendly perspective and that all the features and requirements come from real user needs. From communication buses to connectors position, all have been designed to meet user expectations while complying with the restrictive technical requirements imposed.

The completion of this project required a complete perspective of all the related elements and participants who participated in an engineering project. Every decision taken in a state was the result of the previous decisions made and had a direct impact on the following decisions. This was the reason why it was crucial to have a complete picture of the project, the requirements, and the goals.

The development, carried out in the academic year 2021-2022, had serious problems and delays with respect to the global component crisis that the industry has been suffering during the last years. Some hardware blocks were redesigned more than three times due to quick changes in component stock. Therefore, some of the components already mounted on prototype boards are not the ones chosen during the development process but compatible ones. In addition, manufacturing was delayed for several months while some key components were in storage. This is the reason why it was not possible to do more than one iteration in hardware development, adding more pressure in the design process: it was only possible to make one prototype, and it had to work.

Overall, the project has been completed successfully meeting the proposed goals and requirements. As a result, a working prototype capable of controlling motors

with a complete complex and fully configurable control loop was produced. This prototype is interfaced with the widely used ROS middleware, which provides complete interoperability with other systems, hundreds of tools to configure and control the system, and a standard interface with the hardware.

Personally, this project gave me the opportunity to work in a real application system with technical and economical constraints. During this project, I learned to convert general ideas into real and implementable requirements. I finish this project with a deeper understanding of embedded system design and robotics.

## 5.2 Future improvements

During the development and testing processes, new possible requirements, features, and improvements were found. Some of these might be included in future iterations and versions of this project. These improvements are as follows:

- Create new power boards: This project has been designed to easily be moved to other applications by changing the power boards. It is proposed to create boards to test and exploit this feature.
- Create a desktop application: Although the system can be controlled using the application provided in the ROS framework, it is proposed to create a graphical desktop application to manage the prototype in an easy way.
- Improve data structures: It is proposed to find more efficient data structures for the interchange between the user and the application.
- Improve the communication structure: It is also proposed to review the topic communication structure to find ways to configure and control the system. There are many ROS capabilities that can be exploited to achieve this.
- Fix hardware bugs: Some design problems related to component orientation were found during electrical and visual inspections. These problems can be easily fixed in future hardware revisions.

# Bibliography


[1] Anixter. Twisted-pair ethernet: Copper cabling for high-performance networking. URL: https://www.anixter.com/content/dam/Anixter/White%20Papers/12F0009X00-Anixter-Twisted-Pair-Ethernet-WP-ECS-EN.pdf.

[2] Jakub Bartoszek. Legged robots: Ros2, 2022. URL: https://mab-robotics.medium.com/legged-robots-ros2-6051f9c907cd.

[3] Philip Beard. Application note regarding h bridge design and operation, 2014. URL: https://www.egr.msu.edu/classes/ece480/capstone/fall14/group07/PDFs/Application%20Note%20Regarding%20H%20Bridge%20Design%20and%20Operation.pdf.

[4] Teresa Blas Martín and Ana Fernández Serrano. Generador eléctrico, 2021. URL: https://www2.montes.upm.es/dptos/digfa/cfisica/magnet/generador.html.

[5] Canonical. What is ros, 2022. URL: https://ubuntu.com/robotics/what-is-ros.

[6] Hewlett-Packard Compaq and Lucent Intel. Universal serial bus specification revision 2.0.

[7] Nick Connor. What is thermal resistance – thermal resistivity – definition, 2019. URL: https://www.thermal-engineering.org/what-is-thermal-resistance-thermal-resistivity-definition/.

[8] OKAWA Electric Design. Sallen-key low-pass filter design tool, 2022. URL: http://sim.okawa-denshi.jp/en/OPseikiLowkeisan.htm.

[9] Reference designer. Transmission line, 2022. URL: http://referencedesigner.com/books/si/Transmission_Line.php.

[10] Electrical4U. Control systems: What are they? (open-loop & closed-loop control system examples), 2020. URL: https://www.electrical4u.com/control-system-closed-loop-open-loop-control-system/.

[11] C. S. S. Electronics. Can bus explained - a simple intro. *CSS Electronics*, 2021. URL: https://www.csselectronics.com/pages/can-bus-simple-intro-tutorial.

[12] Institute of Electrical Engineers and Electronics. Ieee standard for ethernet. *IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015)*, pages 1–5600, 2018. doi:10.1109/IEEESTD.2018.8457469.

[13] Xiaocong Fan. *Real-time embedded systems : design principles and engineering practices*. Elsevier, Oxford, U.K., 2015.

[14] Fiware. Getting started with micro-ros: Core and advanced tutorials, 2020. URL: https://www.fiware.org/2020/06/16/getting-started-with-micro-ros-core-and-advanced-tutorials/.

[15] FreeRTOS.org. The freertos™ kernel, 2021. URL: https://www.freertos.org/RTOS.html.

[16] DDS Fundation. What is dds?, 2021. URL: https://www.dds-foundation.org/what-is-dds-3/.

[17] Sourav Gupta. Understanding impedance matching in pcb design with example and calculation, 2021. URL: https://circuitdigest.com/article/understanding-impedance-matching-in-pcb-design-with-example-and-calculation.

[18] Álvaro Gutiérrez. Laboratorio seco, 2021. URL: http://www.robolabo.etsit.upm.es/asignaturas/seco/apuntes/telelabo/secoStudentsQueueAppManual.pdf.

[19] Jr. Hayt, William H. and John A. Buck. *Engineering electromagnetics*. McGraw-Hill series in electrical engineering. McGraw-Hill, Boston, 6th edition, 2001.

[20] Jessica Hopkins. An introduction to the usb protocol, 2020. URL: https://www.totalphase.com/blog/2020/07/about-the-usb-protocol-common-usb-bus-errors-and-how-to-troubleshoot-them/.

[21] Texas Instruments. Slva057: Undestanding switching power stages in switchmode power supplies, 199. URL: https://www.ti.com/lit/an/slva057/slva057.pdf?ts=1646470356420.

[22] Sean Keane. Microsoft is mining the xbox 360 'red ring' controversy for profit, and that's not cool, 2021. URL: https://www.cnet.com/tech/gaming/microsoft-is-mining-xbox-360-red-ring-controversy-for-profit-and-thats-not-cool/.

[23] Kella Knac. Single-ended switching versus differential signaling, 2020. URL: https://resources.altium.com/p/single-ended-switching-versus-differential-signaling-0.

[24] Phil Koopman. A case study of toyota unintended acceleration and software safety, 2014. URL: https://users.ece.cmu.edu/~koopman/pubs/koopman14_toyota_ua_slides.pdf.

[25] Ralph Lange. Micro-ros – bringing the most popular robotics middleware onto tiny microcontrollers, 2021. URL: https://www.bosch.com/stories/bringing-robotics-middleware-onto-tiny-microcontrollers/.

[26] Edward A. Lee and Sanjit A. Seshia. *Introduction to embedded systems : a cyber-physical systems approach*. MIT Press, Cambridge, Massachuetts, second edition. edition, 2017.

[27] Smeeta Maheriya and Priyam Parikh. A review: Modelling of brushed dc motor and various type of control methods. *Journal for Research— Volume 01— Issue 12 — February 2016 ISSN: 2395-7549*, 12:18–23, 2016.

[28] Abhijit Majumdar, Apurba Das, R. Alagirusamy, and V. K. Kothari. *Process Control in Textile Manufacturing*, pages 475–492. Woodhead Publishing, 2013. URL: https://www.sciencedirect.com/science/article/pii/B9780857090270500229, doi:https://doi.org/10.1533/9780857095633.backmatter.

[29] Ken Marasco. How to successfully apply low dropout regulators . Report, Analog, 2010. URL: https://www.analog.com/media/en/technical-documentation/application-notes/AN-1072.pdf.

[30] University of Michigan. Introduction: Digital controller design, 2020. URL: https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlDigital.

[31] micro ROS. Features and architecture, 2022. URL: https://micro.ros.org/docs/overview/features/.

[32] micro ROS. Micro xrce-dds, 2022. URL: https://micro.ros.org/docs/concepts/middleware/Micro_XRCE-DDS/.

[33] Microchip. Mcp9808: ±0.5°c maximum accuracy digital temperature sensor, 2011. URL: https://www.mouser.mx/datasheet/2/268/MCP3561_2_4_Family_Data_Sheet_DS20006181C-2257924.pdf.

[34] Microchip. Small footprint rmii 10/100 ethernet transceiver with hp auto-mdix support, 2012. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/8720a.pdf.

[35] Microchip. An2434: Interfacing quadrature encoder using ccl with tca and tcb, 2017. URL: http://ww1.microchip.com/downloads/en/AppNotes/00002434A.pdf.

[36] Microchip. Mic4606: 85v full-bridge mosfet drivers with adaptive dead time and shoot-through protection, 2019. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/MIC4606-Data-Sheet-DS20005604D.pdf.

[37] Microchip. Mcp3561: Two/four/eight-channel, 153.6 ksps, low noise 24-bit delta-sigma adcs, 2021. URL: https://www.mouser.mx/datasheet/2/268/MCP3561_2_4_Family_Data_Sheet_DS20006181C-2257924.pdf.

[38] Félix Monasterio-Huelin and Alvaro Gutiérrez. Introducció n a los sistemas electró nicos de control, 2020. URL: http://robolabo.etsit.upm.es/asignaturas/seco/apuntes/2019-2020/introSECO.pdf.

[39] Carmine Noviello. *Mastering STM32*. leanpub, 2018.

[40] NXP. Mc33hb2001: 10 a h-bridge, spi programmable brushed dc motor driver, 2020. URL: https://www.nxp.com/docs/en/data-sheet/MC33HB2001.pdf.

[41] NXP. i.mx rt1050 crossover mcu with arm cortex-m7 core. Web page, NXP, 2022. URL: https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/i-mx-rt-crossover-mcus/i-mx-rt1050-crossover-mcu-with-arm-cortex-m7-core:i.MX-RT1050.

[42] Jacob Pedersen. *Model Based and Robust Control Techniques for Internal Combustion Engine Throttle Valves*. Thesis, University of Southern Denmark, 2013.

[43] Peterson and Zachariah. Pcb impedance calculator and stackup design in altium designer, 2021. URL: https://resources.altium.com/p/pcb-stackup-impedance-calculator.

[44] Eric Peña. Uart: A hardware communication protocol understanding universal asynchronous receiver/transmitter, 2020. URL: https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html.

[45] Oxford University Press. Hardware. URL: https://www.oed.com/view/Entry/240257?redirectedFrom=akrasia.

[46] Marius Rangu. Getting emc design right – first time, part 3: High-speed signals & impedance, 2014. URL: https://www.edn.com/getting-emc-design-right-first-time-part-3-high-speed-signals-impedance/.

[47] RLS. What is the difference between cpr and ppr?, 2022. URL: https://www.rls.si/eng/faq/index/show/cat_id/2/faq/59.

[48] Open Robotics. Ros 2 documentation, 2022. URL: https://docs.ros.org/en/foxy/index.html.

[49] Open Robotics. Ros: Tutorials, 2022. URL: https://docs.ros.org/en/foxy/Tutorials.html.

[50] Open Robotics. Understanding ros 2 actions, 2022. URL: https://docs.ros.org/en/foxy/Tutorials/Understanding-ROS2-Actions.html.

[51] Open Robotics. What is ros?, 2022. URL: http://wiki.ros.org/ROS/Introduction.

[52] Kristian Saxrud. Ds-00003197a- differential and single-ended adc, 2019. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Differential-and-Single-Ended-ADC-WhitePaper-DS00003197A.pdf.

[53] Rohde & Schwarz. Qué es uart, 2022. URL: https://www.rohde-schwarz.com/es/productos/test-y-medida/osciloscopios/educational-content/que-es-uart_254524.html.

[54] Simplia. Compact imxrt 1052 : Functional description, 2022. URL: https://simplia-electronics.com/ficha/compact-imxrt-1052/.

[55] SIMSCALE. What is joule heating?, 2021. URL: https://www.simscale.com/docs/simwiki/heat-transfer-thermal-analysis/what-is-joule-heating/.

[56] Grant Maloy Smith. What is can bus (controller area network). *Dewesoft*, 2021. URL: https://dewesoft.com/daq/what-is-can-bus.

[57] Cadence PCB Solutions. Mii and rmii routing guidelines for ethernet, 2019. URL: https://resources.pcb.cadence.com/blog/2019-mii-and-rmii-routing-guidelines-for-ethernet.

[58] CADENCE PCB SOLUTIONS. What are the maximum power output and data transfer rates for the usb standards?, 2020. URL: https://resources.pcb.cadence.com/blog/2020-what-are-the-maximum-power-output-and-data-transfer-rates-for-the-usb-stand

[59] Cadence PCB Solutions. Exploring the difference between single-ended and differential signals, 2021. URL: https://resources.pcb.cadence.com/blog/msa2021-exploring-the-difference-between-single-ended-and-differential-signals.

[60] Typhoon-hil. Can bus protocol. *Typhoon-hil*, 2022. URL: https://www.typhoon-hil.com/documentation/typhoon-hil-software-manual/References/can_bus_protocol.html.

[61] USB. Universal serial bus 3.1 specification (1.0), 2013.

[62] Tolga Özer, Sinan Kivrak, and Yüksel Oğuz. H bridge dc motor driver design and implementation with using dspic30f4011. *International Journal of Innovative Research in Science, Engineering and Technology*, 6:75–83, 2017.

# Appendix A

# Ethical, social, economic and environmental aspects

## A.1 Social impact

Today, nearly all human appliances, from toothbrushes to autonomous cars, include at least one microcontroller that turns it into an embedded system. The way in which this kind of system works has a direct impact on user security and satisfaction. Therefore, the creation and design of control systems is crucial for our society.

This project has been designed to provide powerful tools to engineers and control system designers. These tools can facilitate the modeling, design, and implementation of complex systems in a way that can be applied directly in user applications. This project abstracts complex hardware layers, allowing the engineer to focus more on creating a good control system. It is evident that this improvement increases the quality of consumer electronics and then the security and satisfaction of the user.

## A.2 Economic impact

Control systems are directly related to the robotic industry. During the last decade, exponential growth has been observed in this industry. The technology sector requires more sophisticated control systems to respond to the increasing demand for new products and services.

An authentic revolution in the robotic industry is being experimented due to the introduction of new technologies of autonomous systems and machine learning. All in all, from an economic point of view, the topic of this project can be located at the top of industry interests and investment.

## A.3 Environmental impact

The creation of any electronic system implies a negative environmental impact. Electronic devices use non-recyclable and, in some cases, toxic materials and

fabrication processes that can contaminate the soil and water if they are properly managed. However, this negative impact can be reduced and compensated for.

As shown in previous sections, this project has been designed to be modular. The main advantage of this design is that it can be easily extended to other applications without the need to re-make all the blocks, making the system flexible and reusable. The lower hardware requirement compensates for part of the negative impact that the creation of this system generates.

## A.4 Conclusion

The industry demand for control solutions has increased during the last decades. These solutions are applied in consumer electronics and the generation of raw materials in most cases. Therefore, control systems have a crucial impact on the daily life of the human population.

The increasing interest in these types of solutions is generating a rich market niche and ecosystem. In addition, this solution may report economic and environmental benefits if used to improve production processes in industrial applications. Finally, its modular design makes it flexible, reusable, extensible, and scaleable, reducing the environmental impact of its creation.

# Appendix B

# Project budget.

Table B.1: Project budget.

| Human Resources | | | |
|---|---|---|---|
| | **Hours** | **€/h** | **total** |
| Electronic Engineer | 450 | 21 € | 9,450 € |

| Depreciation | | | | |
|---|---|---|---|---|
| | **Price** | **Time of use (years)** | **Lifetime (years)** | **Depreciation** |
| Power Supply | 100.00 € | 30.00% | 5 | 6.00 € |
| Soldering Station | 1,500.00 € | 30.00% | 5 | 90.00 € |
| Oscilloscope | 2,000.00 € | 30.00% | 7 | 85.71 € |
| Digital Multimeter | 50.00 € | 30.00% | 7 | 2.14 € |
| Computer | 1,500.00 € | 60.00% | 5 | 180.00 € |
| Software Licences | 3,000.00 € | 40.00% | 1 | 1,200.00 € |
| Development kits | 500.00 € | 20.00% | 3 | 33.33 € |
| Tools | 500.00 € | 60.00% | 5 | 60.00 € |
| Laboratory material | 200.00 € | 50.00% | 1 | 100.00 € |
| Total | | | | 1,757.19 € |

| Other expenses | | |
|---|---|---|
| | **Percentage** | **total** |
| Facilities and maintenance | - | 550 € |
| Indirect costs | 13% | 228.43 € |
| Total | | 778 € |

| Fabrication expenses | |
|---|---|
| | **Cost** |
| Components | 564.00 € |
| Printed circuits | 293.02 € |
| Assembly | 350.00 € |
| Freight charges | 60.00 € |
| Total | 1,267 € |

| Results | | |
|---|---|---|
| | **Percentage** | **Total** |
| Total before taxes | | 13,252.65 € |
| Benefit margin | 30% | 3,975.79 € |
| Taxes | 21% | 3,617.97 € |
| **Total cost** | | **20,846.41 €** |

# Appendix C

# Manual: How to configure the project environment

## C.1 Install dependencies

The project repository is located on Github and can be downloaded using the following command:

```
git clone https://github.com/Robolabo/COMPAC_MotorControl.git
```

The first step is to install ROS2, the ARM GCC compiler, the micro-ROS agent, and the required dependencies. This process has been automated in a scrip located in the micro-ROS folder. Move to that folder and execute the script using the following commands:

```
cd COMPAC_MotorControl/Software/Firmware/microROS/
sudo -s source setup.sh
```

This script has been developed and tested to install ROS2 foxy in Ubuntu 20.04. Using a different version of Ubuntu may require installing other ROS2 distributions. For doing this, change edit the distribution version parameter in the 'setup.sh' script:

```
#Ubuntu 18.04: dashing, eloquent
#Ubuntu 20.04: foxy, galactic
#Ubuntu 22.04: rolling
export ROS_DISTRO=foxy
```

Some of the installation paths are relative to the user, and, since the script is executed as 'sudo', it is recommended to execute the script again without 'sudo' to verify everything has been installed correctly.

```
source setup.sh
```

On every new terminal, it is necessary to set up the ROS2 environment before starting to use its packages. The environment is set up using the following commands:

```
export ROS_DISTRO=foxy
source /opt/ros/$ROS_DISTRO/setup.bash
source ./motor_data_msg/install/setup.bash
```

Remember that it is necessary to set up the environment in every session. To avoid doing this, you can add the previous lines at the end of the file /-bashrc:

## C.2 Starting XRCE-DDS agent

One of the packages installed using the script 'setup.sh' is the XRCE-DDS agent, which acts as a point of union between ROS2 and micro-ROS. It is necessary to launch this application to start communication with the board. The agent is launched depending on the communication protocol used:

- UDP: The program requires as parameters the UDP port used. The IP and port of this application must be the same as those configured on micro-ROS. In the following example, the agent is starting in UDP mode on port 8888.

```
MicroXRCEAgent udp4 -p 8888
```

- UART: The program requires the device and the baudrate. The baudrate must be the same as that configured in the application. In the following example, the agent is starting in serial mode using the /dev/ttyUSB0 device with baudrate 576000.

```
MicroXRCEAgent serial -b 576000 -D /dev/ttyUSB0
```

- CAN: The program requires the name of the device as a parameter. In the following example, the agent is starting in CAN mode using the can0 device.

```
MicroXRCEAgent can -D can0
```

Now, if everything was done correctly, you can connect the board and see the registration log in the agent application, as shown in Figure C.1.

You can test if topics and services have been correctly registered using the following commands to list them:

```
ros2 service list # List registered services
ros2 topic list # List registered topics
```

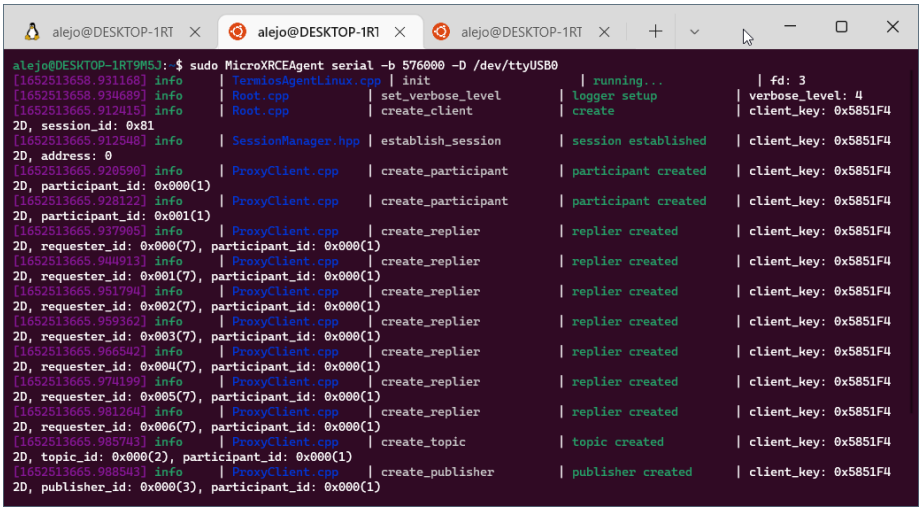



Figure C.1: XRCE-DDS agent register output.

## C.3 Compiling micro-ROS middleware

The generation and compilation of the static library micro-ROS is automated in the 'build' script located in the micro-ROS folder. The execution of this script requires the installation of all ROS2 dependencies (see Section C.1).

The maximum number of publishers, subscribers, services, and the default configuration of the UDP layer are defined in the COLCON_UART.meta, COLCON_UDP.meta and, COLCON_CAN.meta. In the case of the UDP file, it is important to define the agent IP and port by editing the following parameters:

```
...
        "rmw_microxrcedds": {
            "cmake-args": [
                 .....
                "-DRMW_UXRCE_TRANSPORT=udp",
                "-DRMW_UXRCE_DEFAULT_UDP_IP=10.10.10.2",
                "-DRMW_UXRCE_DEFAULT_UDP_PORT=8888"
                 ....
            ]
        }
...
```

The selection of the bus must be done before the compilation. This script allows for three possible options: UDP, SERIAL and CAN. The compilation process is done calling the build script one of the options as shown:

```
source build.sh SERIAL
```

The result of the compilation is in the libmicroros folder. It is also recommended not to delete the firmware and uros_ws folders to avoid generating and compiling all the code again.

## C.4 Compiling and flashing the COMPAC_MotorControl project

The development of this application has been done using the MCUXpresso IDE. This IDE can be downloaded from the NXP website. The first step is to download and install the software.

The next step is to install the Simplia SDK. As shown in Figure C.2, click on the 'installed SDKS' tab and right-click on the list panel. This will show a drop-down menu where it should be selected as 'import archive'. Then it is necessary to select the route of the Simplia SDK zip file.

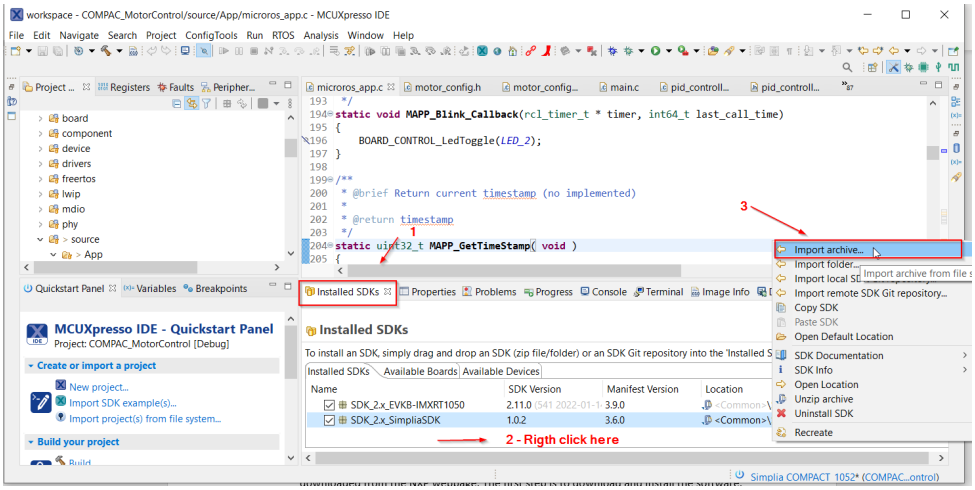



Figure C.2: MCUXpresso SDK instalation.

Now, the project can be imported. Click on 'file → Open Project From File System and then Directory' (see Figure C.3). The project is allocated in the folder Software/Firmware of the repository, choose this folder, and then click on 'Finish' (see Figure C.4). This will load the project.

Once the project is loaded, it can be compiled by clicking the hammer icon and flashed and debugged by clicking on the bug icon (see Figure C.5). Note that it is necessary to generate and compile the micro-ROS library before compiling the application project.

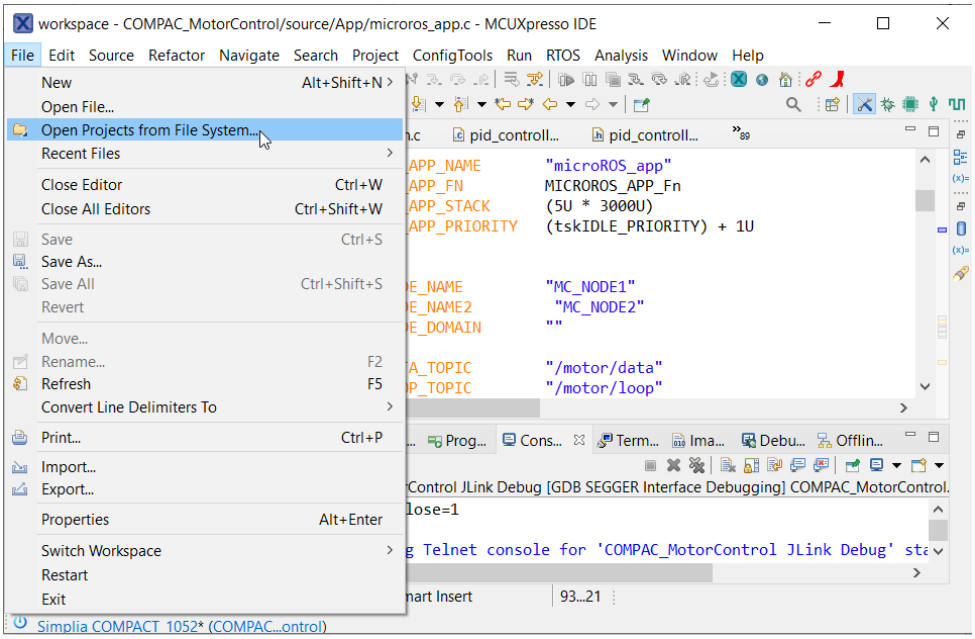



Figure C.3: MCUXpresso project import part 0.

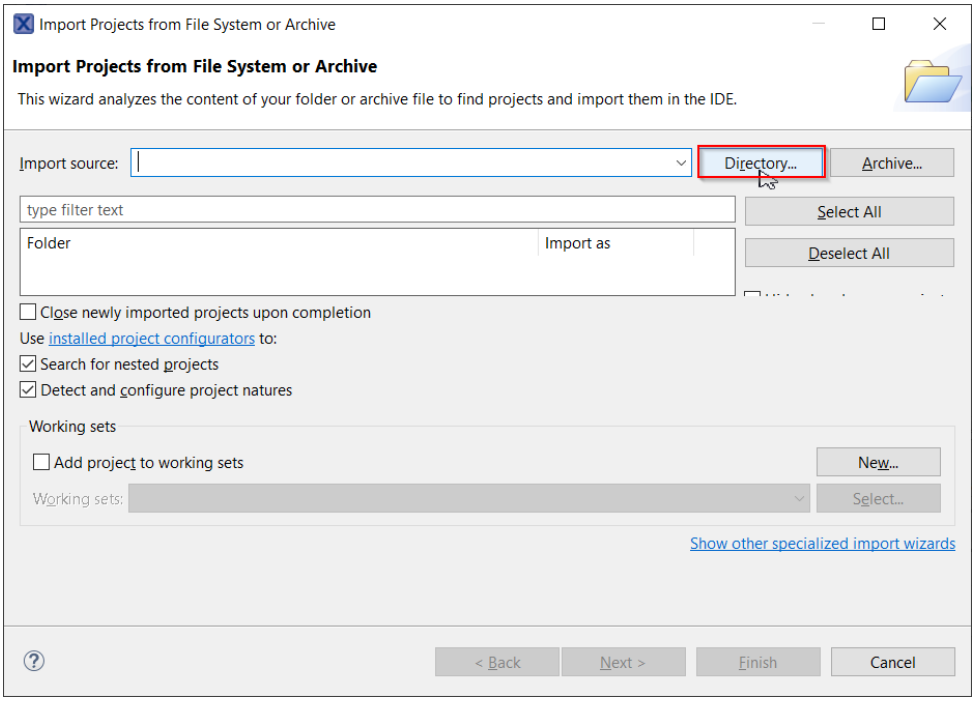



Figure C.4: MCUXpresso project import part 1.

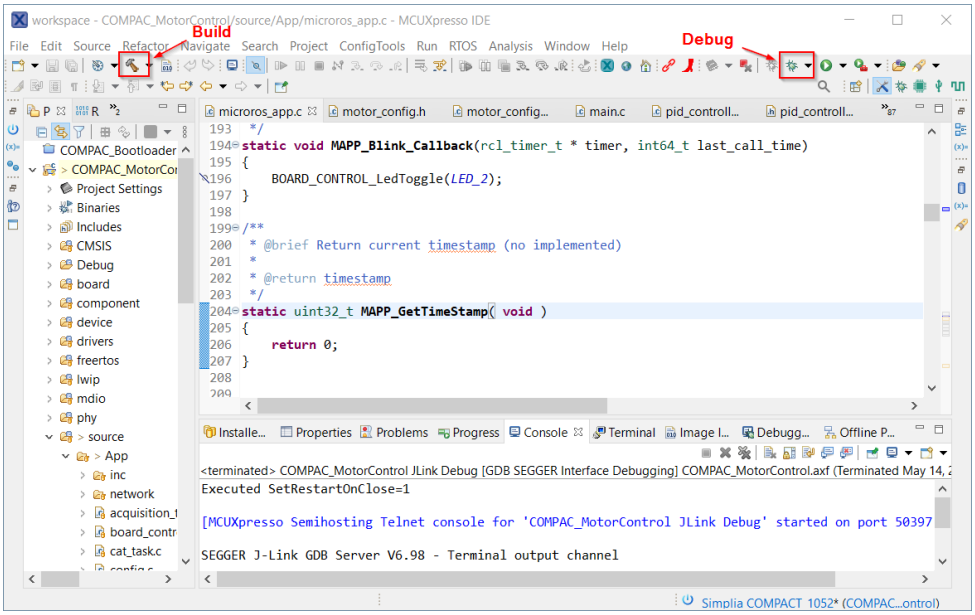



Figure C.5: MCUXpresso compiling and debugging options.

# Appendix D

# ROS message types description.

The following tables describe the data structures created to configure the hardware and extract the data generated by the system.

Table D.1: motor_data_msg/srv/MotorConfig message description.

| Unit | Variable | Description | Values |
|---|---|---|---|
| uint8 | motor_id | Motor identifier | '0' - motor 1<br>'1' - motor 2 |
| uin32 | motor_freq_hz | PWM frequency | Herzt (Hz) |
| uin32 | control_period | Controller period | milliseconds (ms) |
| uin32 | speed_period | Speed measurement period | milliseconds (ms) |
| uin32 | counts_per_rebolution | Reduction and encoder CPR | Pulses |
| uint8 | mode | Operation mode | '0' - Position<br>'1' - Speed<br>'2' - Current |
| uint8 | reference_source | Reference source | '0' - Internal (stimulus)<br>'1' -external |
| float32 | supply_voltage | Motor supply voltage | Volts (V) |

Table D.2: motor_data_msg/srv/MotorData message description.

| Unit | Variable | Description | Values |
|---|---|---|---|
| uint8 | motor_id | Motor identifier | '0' - motor 1<br>'1' - motor 2 |
| uin32 | time_stamp | Control time | Control ticks |
| uint32 | speed | Motor speed | rad/s |
| float32 | position | Relative position | radians |
| float32 | current | Motor current | ampers |
| float32 | control_dir | Direct loop output | - |
| float32 | control_par | Parallel loop output | - |
| float32 | control_ff | Feedforward loop output | - |
| float32 | control_fb | Feedback loop output | - |
| float32 | error | Control signal | - |
| float32 | error_fb | Feedback error | - |

Table D.3: motor_data_msg/srv/Enable message description.

| Unit | Variable | Description | Values |
|---|---|---|---|
| uint8 | motor_id | Motor identifier | '0' - motor 1<br>'1' - motor 2 |
| bool | enable | Value status | True - enable<br>False - disable |

Table D.4: motor_data_msg/srv/Reference message description.

| Unit | Variable | Description | Values |
|---|---|---|---|
| uint8 | motor_id | Motor identifier | '0' - motor 1<br>'1' - motor 2 |
| float32 | reference | Reference value | - |

Table D.5: motor_data_msg/srv/StimulusConfig message config.

| Unit | Variable | Description | Values |
|---|---|---|---|
| uint8 | motor_id | Motor identifier | '0' - motor 1<br>'1' - motor 2 |
| uint8 | stimulus_type | Function type | '0' - Delta<br>'1' - Step<br>'2' - Ramp<br>'3' - Parabola<br>'4' - Sine<br>'5' - Cosine<br>'6' - trapezoid<br>'7' - Exponential |
| float32 | amplitude | Function amplitude | - |
| float32 | Frequency | Sine/cosine frequency | Hertz |
| float32 | t1, t2, t3 | Trapezoid variables | Milliseconds |
| float32 | v1, v2, v3 | Exponential variables | - |

Table D.6: motor_data_msg/srv/LoopConfig Message description.

| Unit | Variable | Description | Values |
|---|---|---|---|
| uint8 | motor_id | Motor identifier | '0' - motor 1<br>'1' - motor 2 |
| uint8 | loop_id | Loop identifier | '0' - Feedfordward<br>'1' - Feedback<br>'2' - Direct<br>'3' - Parallel |
| float32 | kp | Proportional constant | - |
| float32 | ki | Integral constant | - |
| float32 | kd | Derivative constant | - |
| float32 | windup | Windup | - |

Table D.7: motor_data_msg/srv/SensorData message description.

| Unit | Variable | Description | Values |
|---|---|---|---|
| float32 | adc_1 | ADC analog value 1 | Volts (V) |
| float32 | adc_2 | ADC analog value 2 | Volts (V) |
| float32 | adc_3 | ADC analog value 3 | Volts (V) |
| float32 | adc_4 | ADC analog value 4 | Volts (V) |
| float32 | temperature | Board temperature | Celcius degrees |

# Appendix E

# Pin assignment

Table E.1: Pin assignment applied.

| Pin | Peripheral | Signal | Routed pin/signal | Label | Direction |
|---|---|---|---|---|---|
| L10 | ADC1 | IN, 4 | [L10] GPIO_AD_B0_15 | ADC1_IN4 | Input |
| J11 | ADC2 | IN, 5 | [J11] GPIO_AD_B1_00 | ADC2_IN5 | Input |
| M3 | CAN1 | TX | [M3] GPIO_SD_B1_02 | CAN_TX | Output |
| M4 | CAN1 | RX | [M4] GPIO_SD_B1_03 | CAN_RX | Input |
| J1 | ENC1 | PHASE, A | [J1] GPIO_SD_BO_02 | ENC1_A | Input |
| K1 | ENC1 | PHASE, B | [K1] GPIO_SD_B0_03 | ENC1_B | Input |
| H2 | ENC2 | PHASE, A | [H2] GPIO_SD_BO_04 | ENC2_A | Input |
| J2 | ENC2 | PHASE, B | [J2] GPIO_SD_BO_05 | ENC2_B | Input |
| J12 | GPIO1 | gpio_io, 22 | [J12] GPIO_AD_B1_06 | G_EN_1 | Output |
| K10 | GPIO1 | gpio_io, 23 | [K10] GPIO_AD_B1_07 | G_EN_2 | Output |
| L12 | GPIO1 | gpio_io, 20 | [L12] GPIO_AD_B1_04 | G_EN_3 | Output |
| K12 | GPIO1 | gpio_io, 21 | [K12] GPIO_AD_B1_05 | G_EN_4 | Output |
| L13 | GPIO1 | gpio_io, 26 | [L13] GPIO_AD_B1_10 | LED_1 | Output |
| H13 | GPIO1 | gpio_io, 24 | [H13] GPIO_AD_B1_08 | LED_2 | Output |
| B11 | GPIO2 | gpio_io, 17 | [B11] GPIO_B1_01 | SPI4_CS2 | Output |
| D7 | GPIO2 | gpio_io, 00 | [D7] GPIO_BO_00 | GPIO1 | Output |
| B8 | GPIO2 | gpio_io, 05 | [B8] GPIO_BO_05 | GPIO2 | Output |
| D11 | GPIO2 | gpio_io, 19 | [D11] GPIO_B1_03 | SPI4_CS | Output |
| P2 | LPI2C1 | SCL | [P2] GPIO_SD_B1_04 | I2C1_SCK | Input/Output |
| N3 | LPI2C1 | SDA | [N3] GPIO_SD_B1_05 | I2C1_SDA | Input/Output |
| H11 | LPSPI3 | SDI | [H11] GPIO_AD_B1_13 | SPI3_SDI | Not Specified |
| H12 | LPSPI3 | PCS0 | [H12] GPIO_AD_B1_12 | SPI3_CS | Not Specified |
| J14 | LPSPI3 | SCK | [J14] GPIO_AD_B1_15 | SPI3_SCK | Not Specified |
| G12 | LPSPI3 | SDO | [G12] GPIO_AD_B1_14 | SPI3_SDO | Not Specified |
| D8 | LPSPI4 | SCK | [D8] GPIO_BO_03 | SPI4_SCK | Not Specified |
| E8 | LPSPI4 | SDO | [E8] GPIO_BO_02 | SPI4_SDO | Not Specified |
| E7 | LPSPI4 | SDI | [E7] GPIO_BO_01 | SPI4_SDI | Not Specified |
| K14 | LPUART1 | TX | [K14] GPIO_AD_B0_12 | LPUART1_TX | Output |

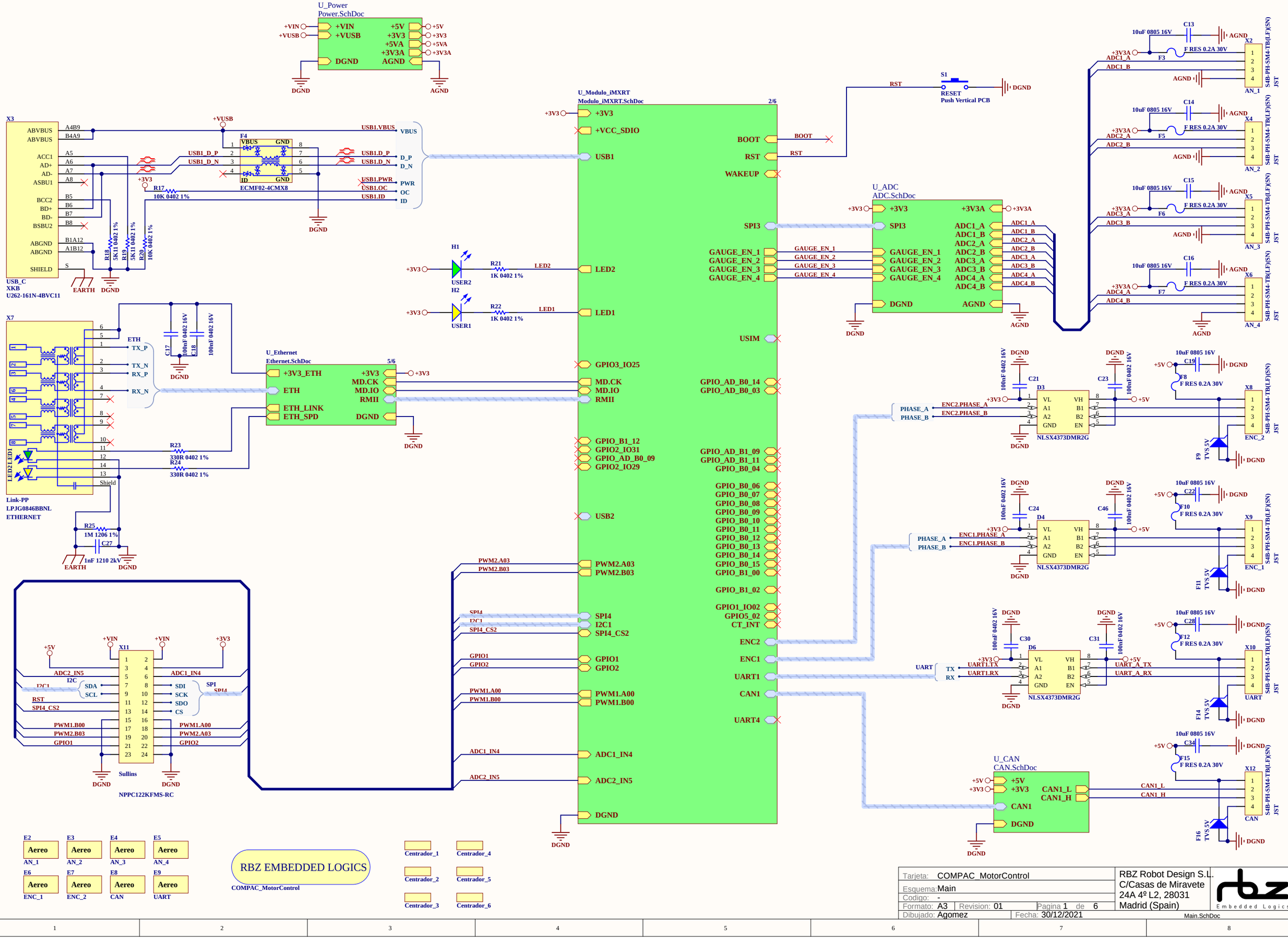
*Continue in the next page*

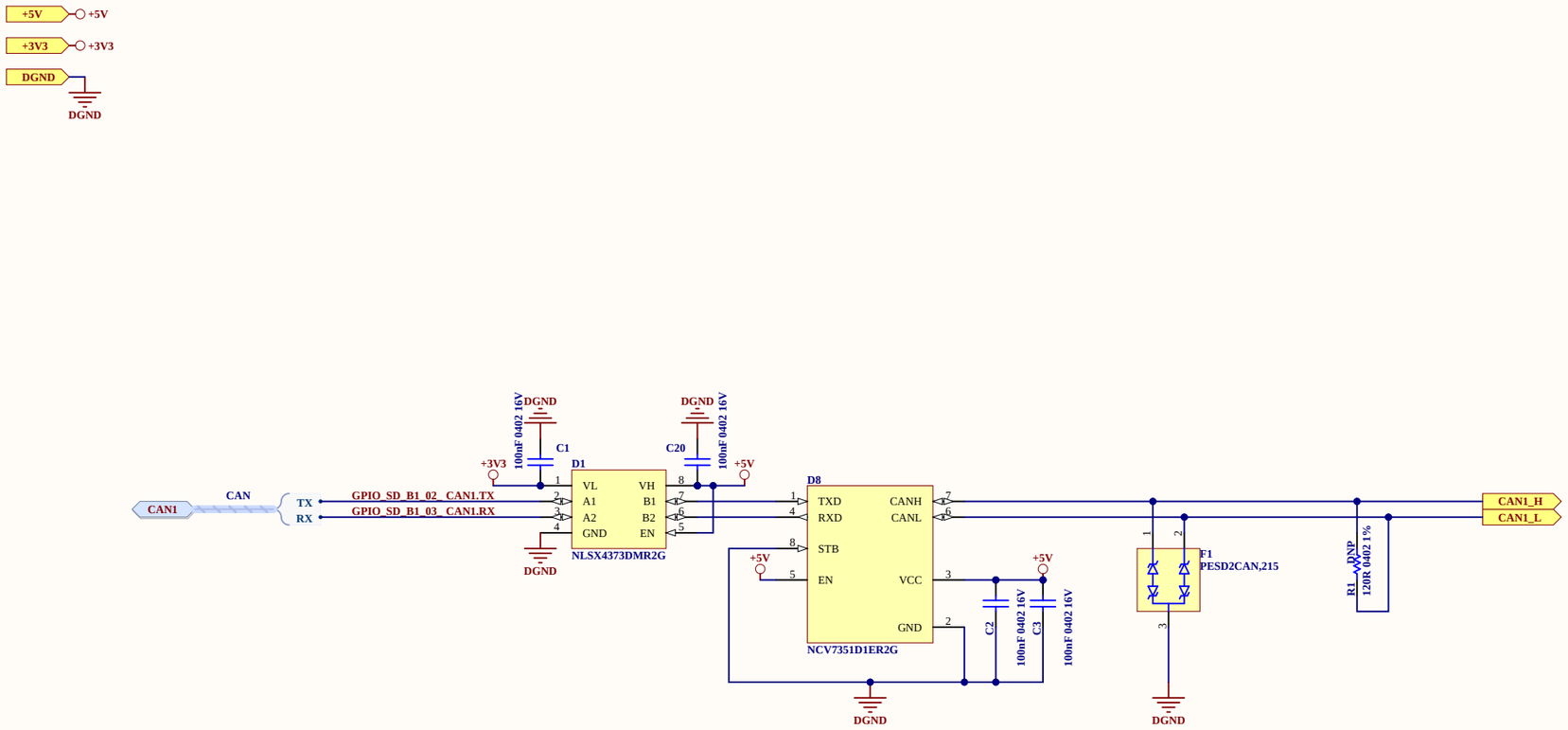
Table E.1 – *Continued from previous page*

| Pin | Peripheral | Signal | Routed pin/signal | Label | Direction |
|---|---|---|---|---|---|
| L14 | LPUART1 | RX | [L14] GPIO_AD_B0_13 | LPUART1_RX | Input |
| J3 | PWM1 | B,0 | [J3] GPIO_SD_B0_01 | PWM1_A | Output |
| J4 | PWM1 | A, 0 | [J4] GPIO_SD_BO_00 | PWM1_B | Output |
| M14 | PWM2 | A, 3 | [M14] GPIO_AD_B0_00 | PWM2_A | Output |
| H10 | PWM2 | B,3 | [H10] GPIO_AD_B0_01 | PWM2_B | Output |
| A7 | ENET | MDC | [A7] GPIO_EMC_40 | ENET_MDC | Output |
| C7 | ENET | MDIO | C7] GPIO_EMC_41 | ENET_MDIO | Input/Output |
| B13 | ENET | REF_CLK | B13] GPIO_B1_10 | ENET_TX_CLK | Not Specified |
| E12 | ENET | RX_DATA, 0 | [E12]GPIO_B1_04 | ENET_RXD0 | Input |
| D12 | ENET | RX_DATA, 1 | (D12] GPIO_B1_05 | ENET_RXD1 | Input |
| C12 | ENET | RX_EN | [C12]GPIO_B1_06 | ENET_CRS_DV | Input |
| C13 | ENET | RX_ER | [C13] GPIO_B1_11 | ENET_RXER | Input |
| B12 | ENET | TX_DATA, 0 | [B12]GPIO_B1_07 | ENET_TXD0 | Output |
| A12 | ENET | TX_DATA, 1 | [A12]GPIO_B1_08 | ENET_TXD1 | Output |
| A13 | ENET | TX_EN | [A13] GPIO_B1_09 | ENET_TXEN | Output |
| L11 | USB1 | OTG1_ID | [L11] GPIO_AD_B1_02 | OTG1_ID | Input |
| M12 | USB1 | OTG1_OC | [M12] GPIO_AD_B1_03 | OTG1_OC | Input |
| K11 | USB1 | OTG1_PWR | [K11] GPIO_AD_B1_01 | OTG1_PWR | Output |

# Appendix F

# Control board schematics.


U_Power
Power.SchDoc
+VIN
+VUSB
DGND
+5V
+3V3
+5VA
+3V3A
AGND
U_Modulo_iMXRT
Modulo_iMXRT.SchDoc
2/6
+3V3
+VCC_SDIO
USB1
WAKEUP
BOOT
RST
SPI3
GAUGE_EN_1
GAUGE_EN_2
GAUGE_EN_3
GAUGE_EN_4
LED2
LED1
USIM
GPIO3_IO25
MD.CK
MD.IO
RMII
GPIO_AD_B0_14
GPIO_AD_B0_03
GPIO_B1_12
GPIO2_IO31
GPIO_AD_B0_09
GPIO2_IO29
GPIO_AD_B1_09
GPIO_AD_B1_11
GPIO_B0_04
USB2
GPIO_B0_06
GPIO_B0_07
GPIO_B0_08
GPIO_B0_09
GPIO_B0_10
GPIO_B0_11
GPIO_B0_12
GPIO_B0_13
GPIO_B0_14
GPIO_B0_15
GPIO_B1_00
PWM2.A03
PWM2.B03
GPIO_B1_02
SPI4
I2C1
SPI4_CS2
GPIO1_IO02
GPIO5_02
CT_INT
GPIO1
GPIO2
ENC2
ENC1
PWM1.A00
PWM1.B00
UART1
CAN1
UART4
ADC1_IN4
ADC2_IN5
DGND
X3
USB_C
XKB
U262-161N-4BVC11
ABVBUS
ACC1
AD+
AD-
ASBU1
BCC2
BD+
BD-
BSBU2
ABGND
SHIELD
EARTH
USB1.VBUS
USB1.D_P
USB1.D_N
USB1.PWR
USB1.OC
USB1.ID
VBUS
D_P
D_N
PWR
OC
ID
F4
ECMF02-4CMX8
R17
10K 0402 1%
R18
5K11 0402 1%
R19
5K11 0402 1%
R20
10K 0402 1%
H1
H2
USER2
USER1
R21
1K 0402 1%
R22
1K 0402 1%
X7
Link-PP
LPJG0846BBNL
ETHERNET
ETH
TX_P
TX_N
RX_P
RX_N
Shield
LED2
LED1
C17
C18
100nF 0402 16V
R23
330R 0402 1%
R24
330R 0402 1%
R25
1M 1206 1%
C27
1nF 1210 2kV
U_Ethernet
Ethernet.SchDoc
5/6
+3V3_ETH
ETH
ETH_LINK
ETH_SPD
X11
NPPC122KFMS-RC
Sullins
ADC2_IN5
ADC1_IN4
I2C
SDA
SCL
SPI
SDI
SCK
SDO
CS
RST
SPI4_CS2
PWM1.B00
PWM2.B03
GPIO1
PWM1.A00
PWM2.A03
GPIO2
S1
RESET
Push Vertical PCB
U_ADC
ADC.SchDoc
+3V3A
ADC1_A
ADC1_B
ADC2_A
ADC2_B
ADC3_A
ADC3_B
ADC4_A
ADC4_B
AGND
C13
C14
C15
C16
10uF 0805 16V
F RES 0.2A 30V
F3
F5
F6
F7
X2
X4
X5
X6
AN_1
AN_2
AN_3
AN_4
S4B-PH-SM4-TB(LF)(SN)
JST
U_CAN
CAN.SchDoc
CAN1_L
CAN1_H
CAN
PHASE_A
PHASE_B
ENC2.PHASE_A
ENC2.PHASE_B
ENC1.PHASE_A
ENC1.PHASE_B
NLSX4373DMR2G
C21
C23
C24
C46
C30
C31
D3
D4
D6
100nF 0402 16V
C19
C22
C28
C34
F8
F10
F12
F15
X8
X9
X10
X12
ENC_2
ENC_1
UART
CAN
F9
F11
F14
F16
TVS 5V
UART
UART_A_TX
UART_A_RX
UART1.TX
UART1.RX
E2
E3
E4
E5
E6
E7
E8
E9
Aereo
AN_1
AN_2
AN_3
AN_4
ENC_1
ENC_2
CAN
UART
RBZ EMBEDDED LOGICS
COMPAC_MotorControl
Centrador_1
Centrador_2
Centrador_3
Centrador_4
Centrador_5
Centrador_6
Tarjeta: COMPAC_MotorControl
Esquema: Main
Codigo: -
Formato: A3
Revision: 01
Pagina 1 de 6
Dibujado: Agomez
Fecha: 30/12/2021
RBZ Robot Design S.L.
C/Casas de Miravete
24A 4º L2, 28031
Madrid (Spain)
Embedded Logics
Main.SchDoc




| | | | |
|---|---|---|---|
| Tarjeta: COMPAC_MotorControl | | | RBZ Robot Design S.L.<br>C/Casas de Miravete<br>24A 4º L2, 28031<br>Madrid (Spain) |
| Esquema: CAN | | | |
| Codigo: - | | | |
| Formato: A3 | Revision: 01 | Pagina 2 de 6 | |
| Dibujado: Agomez | Fecha: 21/12/2021 | CAN.SchDoc | |

+3V3 +3V3

DGND DGND

**STRAP CONFIG**

- **LED1/REGOFF:**
  Activity Indicator.
  Pulled Low, Internal regulator Enable

- **LED2/INTSEL:**
  Link Speed Indicator
  Pulled Low, REF-CLKO selected

-**PHYAD0:**
Pulled High, Unique address

- **MODE0, MODE1, MODE2**
  111 All Capable, Auto negotiaton Enable

+3V3 — L1 FB120R 1A — +3V3_ETH

C4 100nF 0402 16V, C5 100nF 0402 16V, C47 100nF 0402 16V, C6 100nF 0402 16V, C7 100nF 0402 16V, C8 10uF 0805 16V — DGND

C9 1uF 0402 6V3, C10 470pF 0402 50V — DGND

R2 49R9 0402 1%, R3 49R9 0402 1%, R4 49R9 0402 1%, R5 49R9 0402 1%

**D2**
**LAN8720AI-CP**

VDDIO (9), VDD1A (19), VDD2A (1), VDDCR (6)

| | | | |
|---|---|---|---|
| MDIO | 12 | TXP | 21 |
| MDC | 13 | TXN | 20 |
| RXD0/MODE0 | 8 | RXP | 23 |
| RXD1/MODE1 | 7 | RXN | 22 |
| INT/REFCLKO | 14 | LED1/REGOFF | 3 |
| RXER/PHYAD0 | 10 | LED2/INTSEL | 2 |
| TXEN | 16 | RBIAS | 24 |
| TXD0 | 17 | | |
| TXD1 | 18 | | |
| CRS_DV/MODE2 | 11 | | |
| RST | 15 | | |
| XTAL1/CLKIN | 5 | | |
| XTAL2 | 4 | | |
| E-PAD | 25 | | |

+3V3 — R6 10K 0402 1%, R7 10K 0402 1%, R8 10K 0402 1%, R9 33R 0402 1%, R10 10K 0402 1%, R11 10K 0402 1%, R12 10K 0402 1%

MD.IO — MD.IO
MD.CK — MD.CK

RMII:
- RX_D0 — RMII.RX_D0
- RX_D1 — RMII.RX_D1
- TX_CLK — RMII.TX_CLK
- RX_ER — RMII.RX_ER
- TX_EN — RMII.TX_EN
- TX_D0 — RMII.TX_D0
- TX_D1 — RMII.TX_D1
- RX_EN — RMII.RX_EN

R13 1M 0402 1% — G1 25MHz — TP1 — C11 27pF 0402 50V, 25MHZ 16pF, C12 27pF 0402 50V — DGND

R14 12K1 0402 1%, R15 10K 0402 1%, R16 10K 0402 1% — DGND

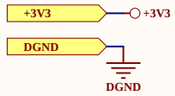
ETH.TX_P, ETH.TX_N, ETH.RX_P, ETH.RX_N — ETH (TX_P, TX_N, RX_P, RX_N)

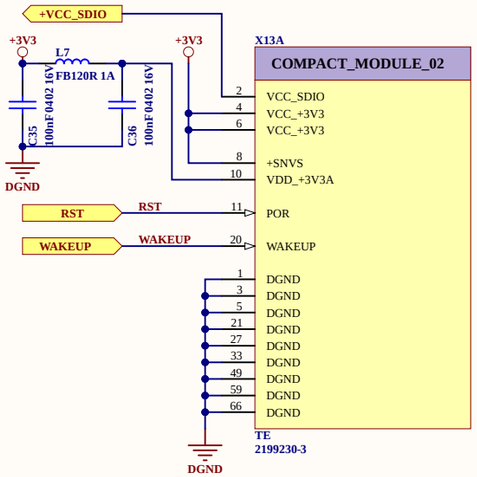
ETH_LINK, ETH_SPD

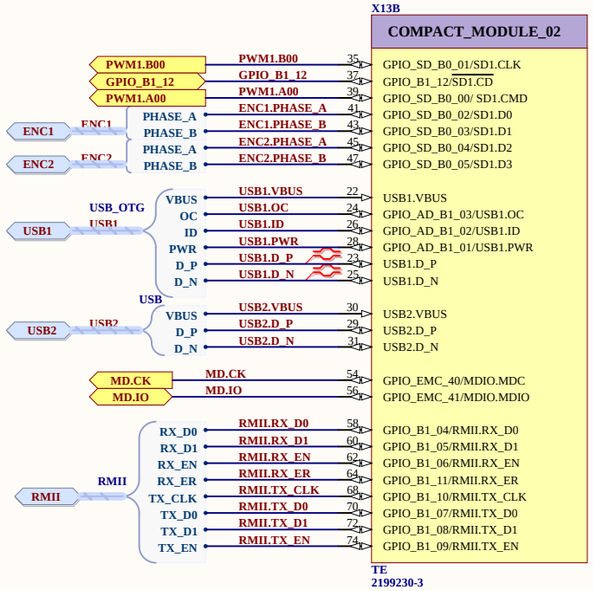
F2 IP4220CZ6,125 — DGND

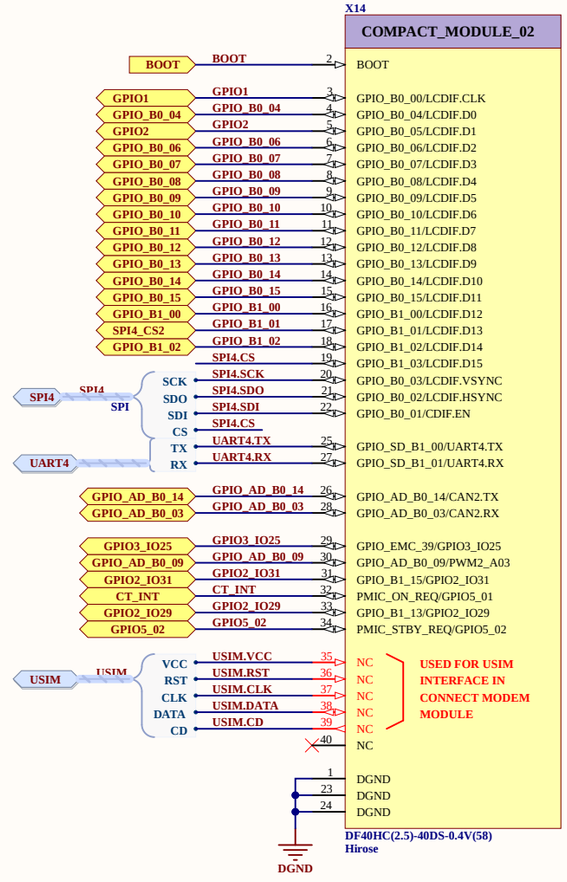
| | |
|---|---|
| Tarjeta: | COMPAC_MotorControl |
| Esquema: | Ethernet |
| Codigo: | 01 |
| Formato: | A3 |
| Revision: | - |
| Pagina | 3 de 6 |
| Dibujado: | Agomez |
| Fecha: | 30/12/2021 |
| File: | Ethernet.SchDoc |

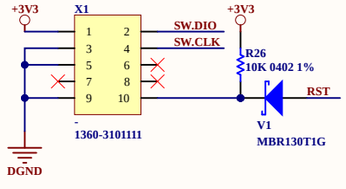
RBZ Robot Design S.L.
C/Casas de Miravete
24A 4º L2, 28031
Madrid (Spain)
Embedded Logics



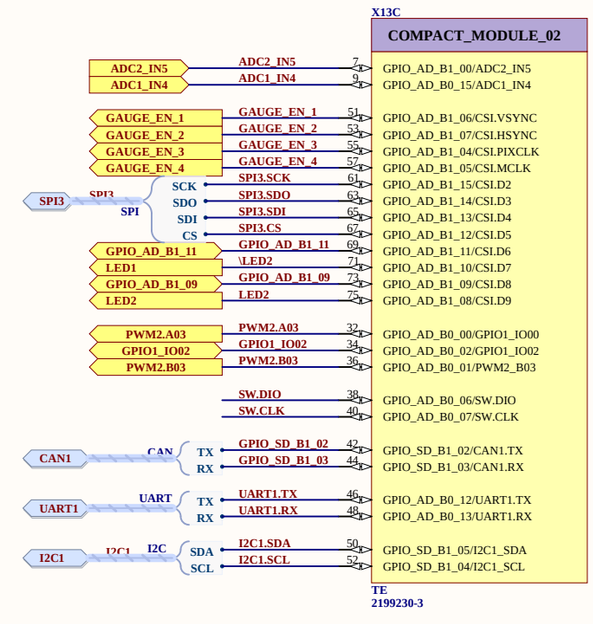





E1
RND STANDOFF M2.5X0.45 STEEL 2.5MM




| | |
|---|---|
| Tarjeta: | COMPAC_MotorControl |
| Esquema: | Modulo_iMXRT |
| Codigo: | - |
| Formato: | A3 |
| Revision: | 01 |
| Pagina | 4 de 6 |
| Dibujado: | Agomez |
| Fecha: | 30/12/2021 |

RBZ Robot Design S.L.
C/Casas de Miravete
24A 4º L2, 28031
Madrid (Spain)

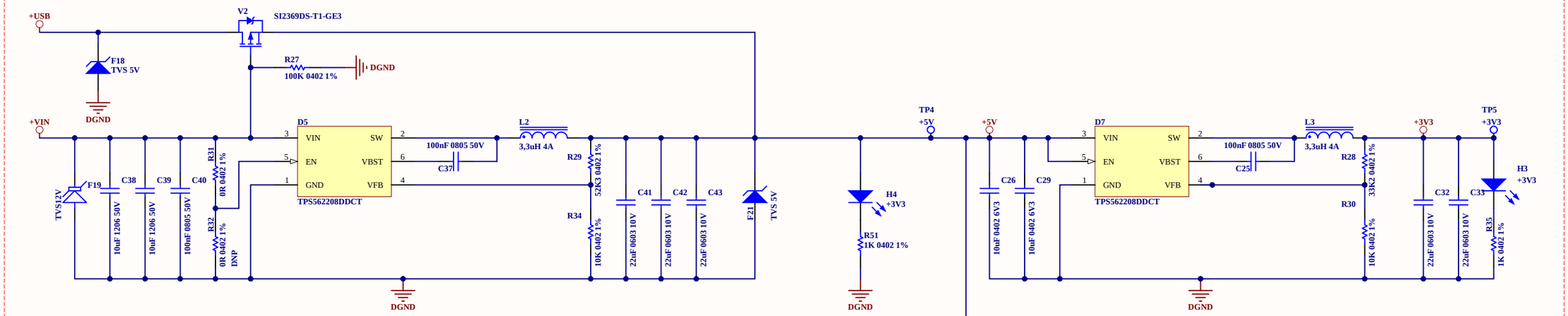
Embedded Logics

Modulo_iMXRT.SchDoc

+VIN +VIN
+VUSB +USB
AGND AGND
DGND DGND

+5V +5V
+3V3 +3V3
+5VA +5VA
+3V3A +3V3A

POWER

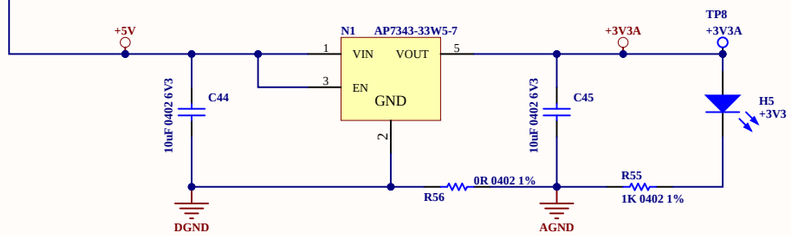



**Potencia entregada máxima**

| Entrada | Tensión | Corriente | Potencia |
|---|---|---|---|
| +Vusb | +5V | 1.5A | 7.25W |
| +Vin | +12V | 625mA | 7.5W |

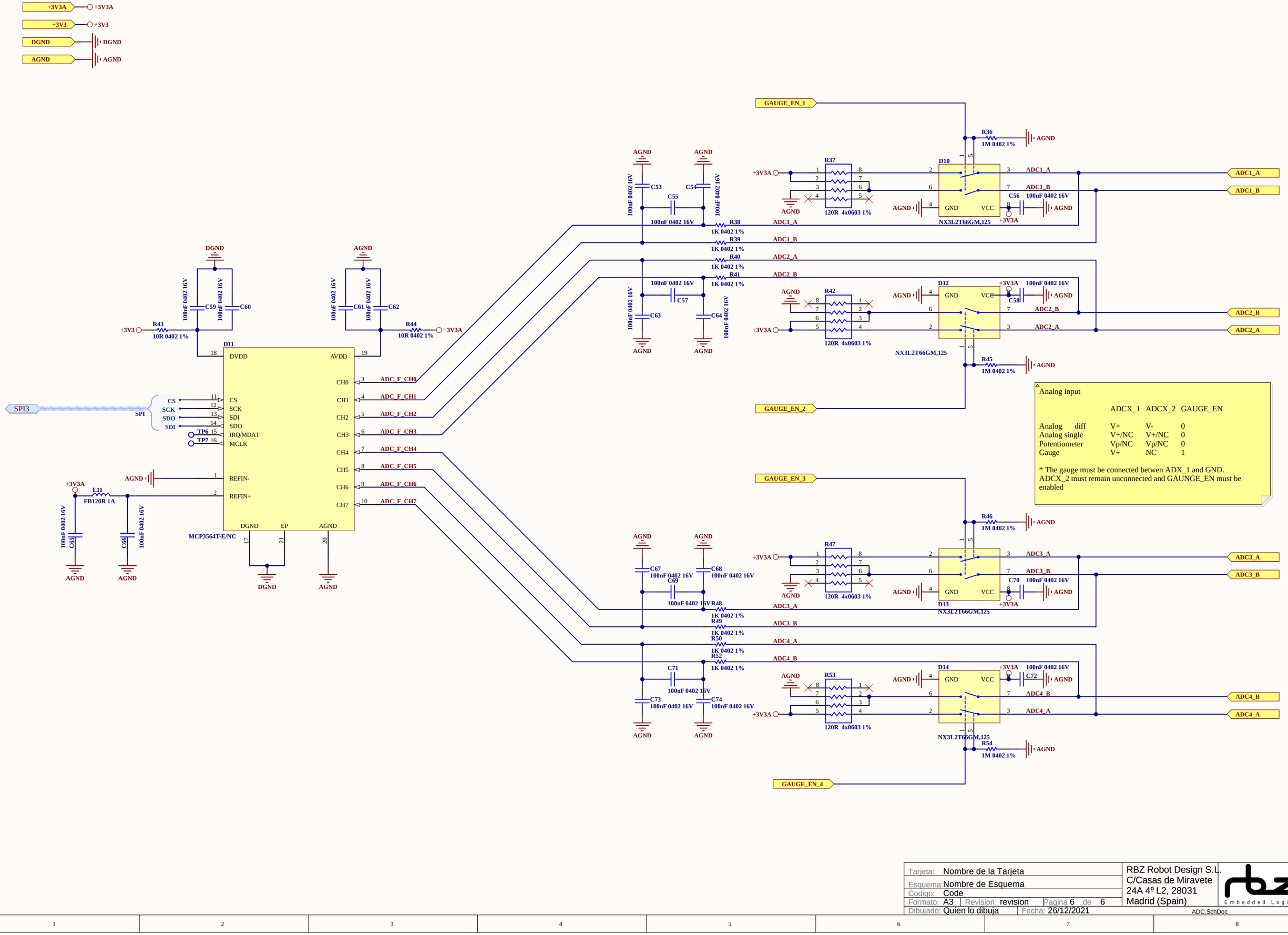
**Potencia consumida máxima**

| Fuente | Corriente | Potencia |
|---|---|---|
| +3.3V | 300mA | 990mW |
| +5V | 900mA | 4.5W |
| +3.3A | 400mA | 2W |
| Total | 1600mA | 7.5W |




| | |
|---|---|
| Tarjeta: | COMPAC_MotorControl |
| Esquema: | Power |
| Codigo: | - |
| Formato: | A3 |
| Revision: | 01 |
| Pagina | 5 de 6 |
| Dibujado: | Agomez |
| Fecha: | 21/12/2021 |
| File: | Power.SchDoc |

RBZ Robot Design S.L.
C/Casas de Miravete
24A 4º L2, 28031
Madrid (Spain)
Embedded Logics

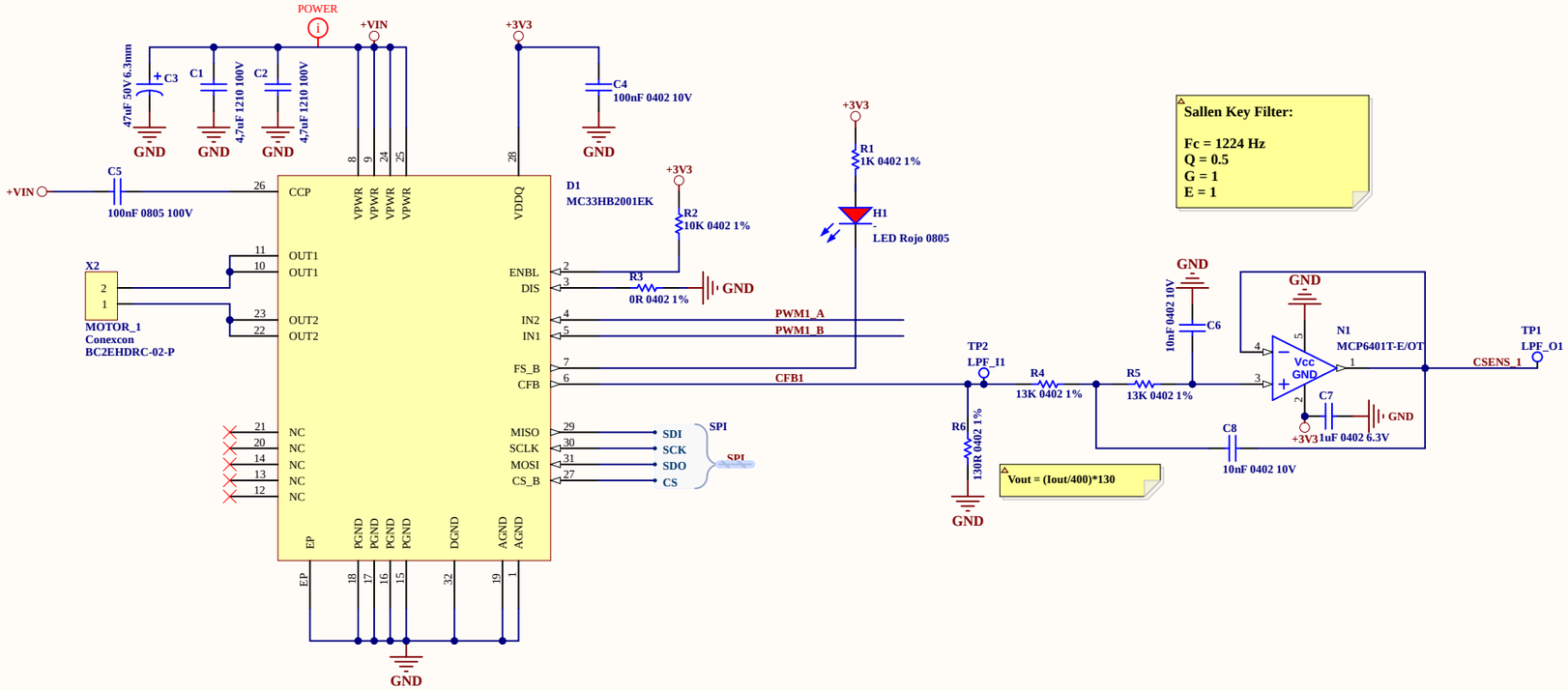


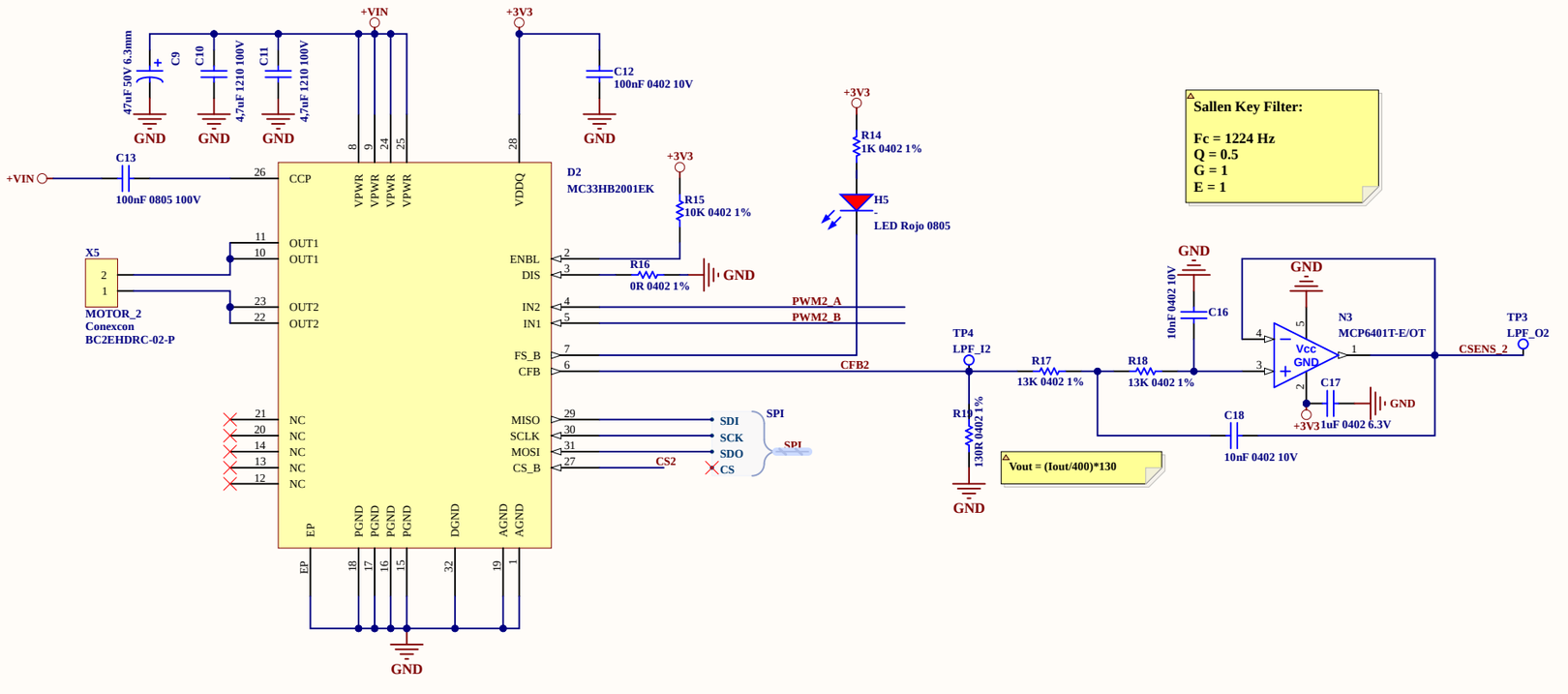

| | | | |
|---|---|---|---|
| Tarjeta: Nombre de la Tarjeta | | | RBZ Robot Design S.L. |
| Esquema: Nombre de Esquema | | | C/Casas de Miravete |
| Codigo: Code | | | 24A 4º L2, 28031 |
| Formato: A3 | Revision: revision | Pagina 6 de 6 | Madrid (Spain) |
| Dibujado: Quien lo dibuja | Fecha: 26/12/2021 | | Embedded Logics |

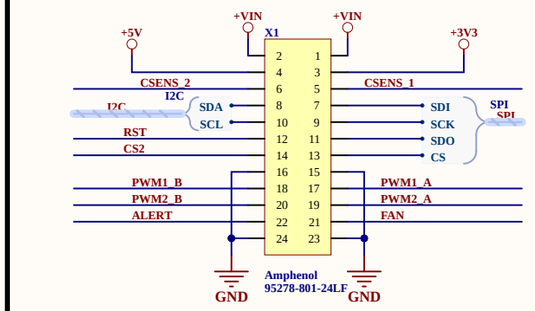
ADC.SchDoc

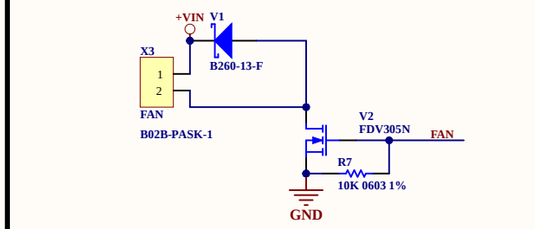
# Appendix G

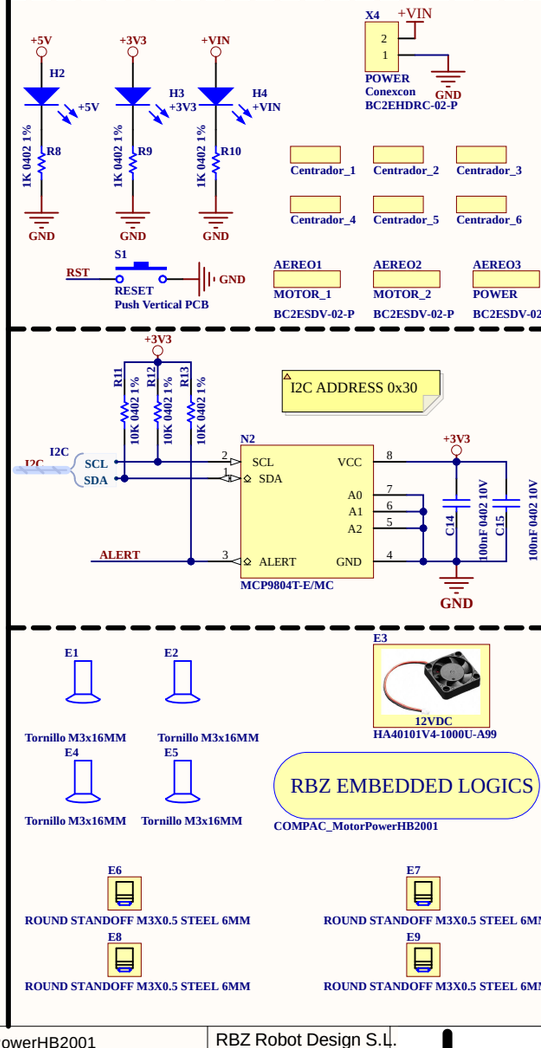
# Motor HB2001 board schematics.



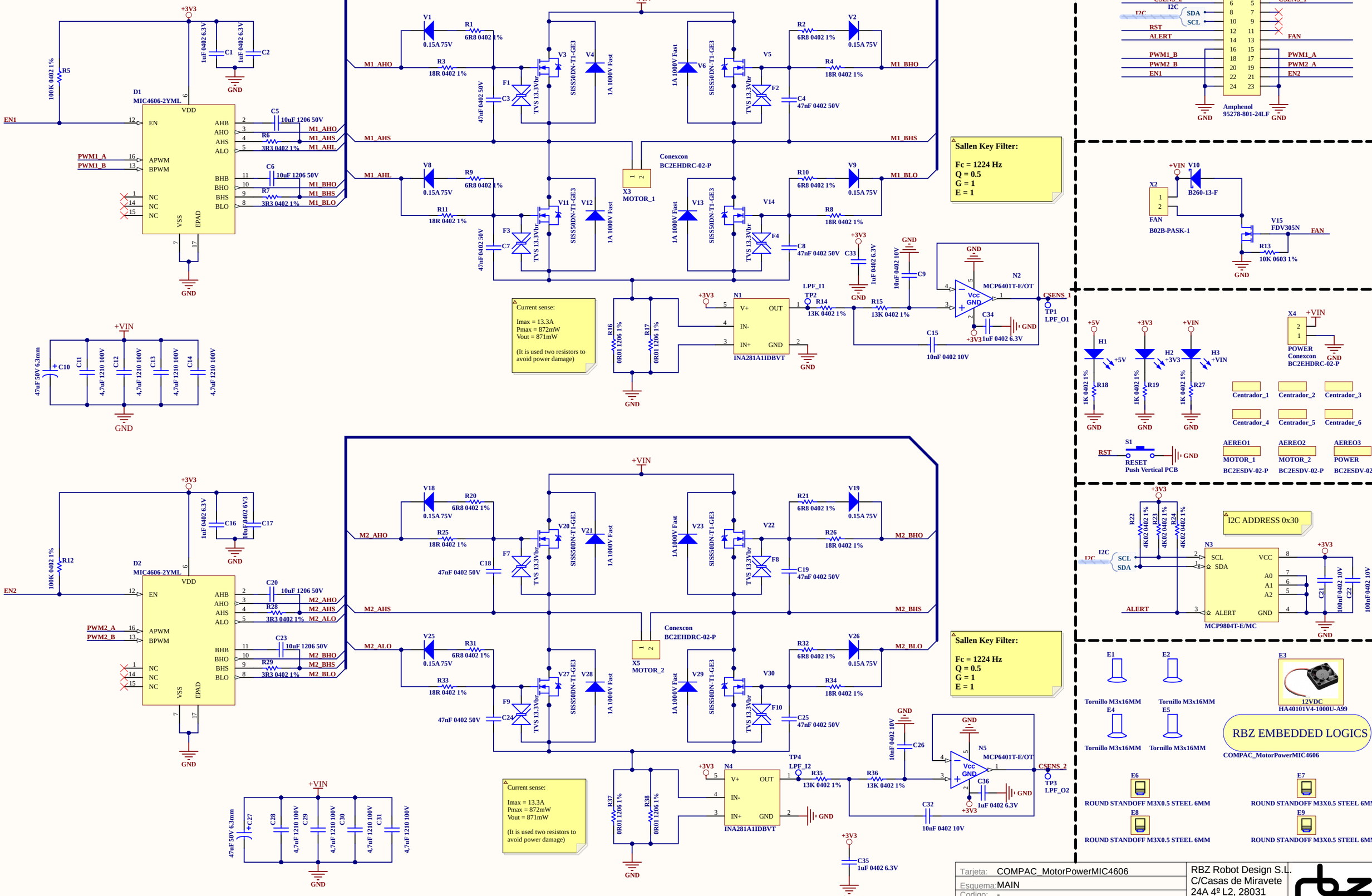




| | | | |
|---|---|---|---|
| Tarjeta: COMPAC_MotorPowerHB2001 | | | RBZ Robot Design S.L. C/Casas de Miravete 24A 4º L2, 28031 Madrid (Spain) |
| Esquema: MAIN | | | |
| Codigo: - | | | |
| Formato: A3 | Revision: 01 | Pagina 1 de 1 | |
| Dibujado: Agomez | Fecha: 30/12/2021 | | MAIN.SchDoc |

# Appendix H

# Motor MIC4606 board schematics.




| | |
|---|---|
| Tarjeta: | COMPAC_MotorPowerMIC4606 |
| Esquema: | MAIN |
| Codigo: | - |
| Formato: | A3 |
| Revision: | 01 |
| Pagina | 1 de 1 |
| Dibujado: | Agomez |
| Fecha: | 30/12/2021 |

RBZ Robot Design S.L.
C/Casas de Miravete
24A 4º L2, 28031
Madrid (Spain)

MAIN.SchDoc

# Appendix I

# Bill of materials

Table I.1: Bill of materials for the COMPAC_MotorControl board.

| Designator | Quantity | Manufacturer | Reference | Value | |
|---|---|---|---|---|---|
| A1 | 1 | Simplia | - | | |
| C1, C2, C3, C4, C5, C6, C7, C17, C18, C20, C21, C23, C24, C30, C31, C35, C36, C46, C47, C53, C54, C55, C56, C57, C58, C59, C60, C61, C62, C63, C64, C65, C66, C67, C68, C69, C70, C71, C72, C73, C74 | 41 | Samsung | CL05B104K O5NNNC | 100nF 16V | 0402 |
| C8, C13, C14, C15, C16, C19, C22, C28, C34 | 9 | Murata | GRM21BR61 C106KE15 | 10uF 16V | 0805 |
| C9 | 1 | Murata | GRM153R60 J105ME95D | 1uF 6V3 | 0402 |
| C10 | 1 | Yageo | CC0402KRX 7R9BB471 | 470pF 50V | 0402 |
| C11, C12 | 2 | Murata | GCM1555C1 H270JA16D | 27pF 50V | 0402 |
| C25, C37, C40 | 3 | Kemet | C0805C104 M5RACTU | 100nF 50V | 0805 |

*Continue in the next page*

Table I.1 – *Continued from previous page*

| Designator | Quantity | Manufacturer | Reference | Value |
|---|---|---|---|---|
| C26, C29, C44, C45 | 4 | Murata | GRJ155R60J106ME11D | 10uF 6V3 0402 |
| C27 | 1 | Johanson | 202S41W102KV4E | 1nF 2kV 1210 |
| C32, C33, C41, C42, C43 | 5 | Murata | GRM188R61A226ME15D | 22uF 10V 0603 |
| C38, C39 | 2 | Samsung | CL31A106KBHNNNE | 10uF 50V 1206 |
| D1, D3, D4, D6 | 4 | ON | NLSX4373DMR2G | |
| D2 | 1 | Microchip | LAN8720AI-CP | |
| D5, D7 | 2 | TI | TPS562208DDCT | |
| D8 | 1 | ON | NCV7351D1ER2G | |
| D10, D12, D13, D14 | 4 | NXP | NX3L2T66GM,125 | |
| D11 | 1 | Microchip | MCP3564T-E/NC | MCP3564T-E/NC |
| E1 | 1 | Wurth | 9774025151R | |
| E2, E3, E4, E5, E6, E7, E8, E9 | 8 | JST | PHR-4 | |
| F1 | 1 | Nexperia | PESD2CAN,215 | |
| F2 | 1 | NXP | IP4220CZ6,125 | |
| F3, F5, F6, F7, F8, F10, F12, F15 | 8 | Bel Fuse | 0ZCJ0020FF2E | F RES 0.2A 30V |
| F4 | 1 | ST | ECMF02-4CMX8 | |
| F9, F11, F14, F16, F18, F21 | 6 | Littelfuse | SMF5.0A | |
| F19 | 1 | Littelfuse | SMBJ12A | |
| G1 | 1 | CTS | ECS-250-18-33-AGM-TR | 25MHZ 18pF |
| H1 | 1 | Kingbright | APT2012SGC | |
| H2 | 1 | Kingbright | APT2012SYCK | |
| H3, H4, H5 | 3 | Kingbright | APT1608SRCPRV | |
| L1, L7, L11 | 3 | Kemet | Z0402C121APMST | FB120R 1A |
| L2, L3 | 2 | Taiyo Yuden | NRS5040T3R3NMGJ | 3,3uH 4A |
| N1 | 1 | DI | AP7343-33W5-7 | |

*Continue in the next page*

Table I.1 – *Continued from previous page*

| Designator | Quantity | Manufacturer | Reference | Value |
|---|---|---|---|---|
| R1 | 1 | Panasonic | ERJ-2RKF1200X | 120R 0402 1% |
| R2, R3, R4, R5 | 4 | Panasonic | ERJ-2RKF49R9X | 49R9 0402 1% |
| R6, R7, R8, R10, R11, R12, R15, R16, R17, R20, R26, R30, R34 | 13 | Vishay | CRCW0402 10K0FKEDC | 10K 0402 1% |
| R9 | 1 | Panasonic | ERJ-2RKF33R0X | 33R 0402 1% |
| R13, R36, R45, R46, R54 | 5 | Panasonic | ERJ-2RKF1004X | 1M 0402 1% |
| R14 | 1 | Panasonic | ERJ-2RKF1212X | 12K1 0402 1% |
| R18, R19 | 2 | Yageo | RC0402FR-075K11L | 5K11 0402 1% |
| R21, R22, R35, R38, R39, R40, R41, R48, R49, R50, R51, R52, R55 | 13 | Vishay | CRCW0402 1K00FKED | 1K 0402 1% |
| R23, R24 | 2 | Panasonic | ERJ-2RKF3300X | 330R 0402 1% |
| R25 | 1 | Vishay | CRCW12061 M00FKEA | 1M 1206 1% |
| R27 | 1 | Panasonic | ERJ-2RKF1003X | 100K 0402 1% |
| R28 | 1 | Panasonic | ERJ-2RKF3322X | 33K2 0402 1% |
| R29 | 1 | Yageo | AC0402FR-0752K3L | 52K3 0402 1% |
| R31, R32, R56 | 3 | Panasonic | ERJ-2GE0R00X | 0R 0402 1% |
| R37, R42, R47, R53 | 4 | Yageo | YC164-FR-07120RL | 120R 4x0603 1% |
| R43, R44 | 2 | Yageo | RC0402FR-0710RL | 10R 0402 1% |
| S1 | 1 | Cannon, ITT | KSR231GLFS | |

*Continue in the next page*

Table I.1 – *Continued from previous page*

| Designator | Quantity | Manufacturer | Reference | Value |
|---|---|---|---|---|
| V1 | 1 | ON Semi | MBR130T1G | Schottky 30V 1A |
| V2 | 1 | Vishay | SI2369DS-T1-GE3 | |
| X1 | 1 | Conexcon | 1360-3101111 | |
| X2, X4, X5, X6, X8, X9, X10, X12 | 8 | JST | S4B-PH-SM4-TB(LF)(SN) | |
| X3 | 1 | XKB | U262-161N-4BVC11 | |
| X7 | 1 | Link-PP | LPJG0846BBNL | |
| X11 | 1 | Sullins | NPPC122KFMS-RC | |
| X13 | 1 | TE | 2199230-3 | |
| X14 | 1 | Hirose | DF40HC(2.5)-40DS-0.4V(58) | |

Table I.2: Bill of materials for the COMPAC_HB2001 board.

| Designator | Quantity | Manufacturer | Reference | Value |
|---|---|---|---|---|
| AEREO1, AEREO2, AEREO3 | 3 | Phoenix Contact/Conexcon | BC2ESDV-02-P | |
| C1, C2, C10, C11 | 4 | AVX | 12101C475K4T2A | 4,7uF 1210 100V |
| C3, C9 | 2 | Nichicon | UUR1H470 MCL6GS | 47uF 50V 6.3mm |
| C4, C12, C14, C15 | 4 | Samsung | CL05B104K P5NNWC | 100nF 0402 10V |
| C5, C13 | 2 | Samsung | CL21B104 KCFWPNE | 100nF 0805 100V |
| C6, C8, C16, C18 | 4 | KEMET | C0402C103 K8PAC7867 | 10nF 0402 10V |
| C7, C17 | 2 | Yageo | CC0402KR X5R5BB105 | 1uF 0402 6.3V |
| D1, D2 | 2 | NXP Freescale | MC33HB2001EK | |
| E1, E2, E4, E5 | 4 | BOSSARD | 1250779 | |

*Continue in the next page*

Table I.2 – *Continued from previous page*

| Designator | Quantity | Manufacturer | Reference | Value |
|---|---|---|---|---|
| E3 | 1 | Sunon Fans | HA40101V4-1000U-A99 | |
| E6, E7, E8, E9 | 4 | Wurth | 9774060360R | |
| H1, H5 | 2 | Kingbright | APT2012EC | |
| H2, H3, H4 | 3 | Kingbright | APT1608SRCPRV | |
| N1, N3 | 2 | Microchip | MCP6401T-E/OT | |
| N2 | 1 | Microchip | MCP9804T-E/MC | |
| R1, R8, R9, R10, R14 | 5 | Vishay | CRCW04021K00FKED | 1K 0402 1% |
| R2, R11, R12, R13, R15 | 5 | Vishay | CRCW040210K0FKEDC | 10K 0402 1% |
| R3, R16 | 2 | Panasonic | ERJ-2GE0R00X | 0R 0402 1% |
| R4, R5, R17, R18 | 4 | Panasonic | ERJ-2RKF1302X | 13K 0402 1% |
| R6, R19 | 2 | Panasonic | ERJ-2RKF1300X | 130R 0402 1% |
| R7 | 1 | Vishay | CRCW060310K0FKEA | 10K 0603 1% |
| S1 | 1 | Cannon, ITT | KSR231GLFS | |
| V1 | 1 | Diodes | B260-13-F | Schottky 60V 2A |
| V2 | 1 | ON Semi | FDV305N | |
| X1 | 1 | Amphenol | 95278-801-24LF | |
| X2, X4, X5 | 3 | Conexcon | BC2EHDRC-02-P | |
| X3 | 1 | JST | B02B-PASK-1 | |

Table I.3: Bill of materials for the COMPAC_MIC4606 board.

| Designator | Quantity | Manufacturer | Reference | Value |
|---|---|---|---|---|
| AEREO1, AEREO2, AEREO3 | 3 | Phoenix Contact/Conexcon | BC2ESDV-02-P | |
| C1, C2, C16, C33, C34, C35, C36 | 7 | Yageo | CC0402KRX5R5BB105 | 1uF 0402 6.3V |

*Continue in the next page*

Table I.3 – *Continued from previous page*

| Designator | Quantity | Manufacturer | Reference | Value |
|---|---|---|---|---|
| C3, C4, C7, C8, C18, C19, C24, C25 | 8 | Yageo | CC0402KR X7R9BB473 | 47nF 50V 0402 |
| C5, C6, C20, C23 | 4 | Samsung | CL31B106K BHNNNE | 10uF 50V 1206 |
| C9, C15, C26, C32 | 4 | KEMET | C0402C103 K8PAC7867 | 10nF 10V 0402 |
| C10, C27 | 2 | Nichicon | UUR1H470 MCL6GS | 47uF 50V 6.3mm |
| C11, C12, C13, C14, C28, C29, C30, C31 | 8 | AVX | 12101C475 K4T2A | 4,7uF 100V 1210 |
| C17 | 1 | Murata | GRM155R60 J106ME15D | 10uF 6V3 0402 |
| C21, C22 | 2 | Kemet | C0402C104 K8PACTU | 100nF 10V 0402 |
| D1, D2 | 2 | Microchip | MIC4606-2YML | |
| E1, E2, E4, E5 | 4 | BOSSARD | 1250779 | |
| E3 | 1 | Sunon Fans | HA40101V4 -1000U-A99 | |
| E6, E7, E8, E9 | 4 | Wurth | 9774060360R | |
| F1, F2, F3, F4, F7, F8, F9, F10 | 8 | Littelfuse | SMAJ12CA | |
| H1, H2, H3 | 3 | Kingbright | APT1608SRCPRV | |
| N1, N4 | 2 | TI | INA281A1IDBVT | |
| N2, N5 | 2 | Microchip | MCP6401T-E/OT | |
| N3 | 1 | Microchip | MCP9804T-E/MC | |
| R1, R2, R9, R10, R20, R21, R31, R32 | 8 | YAGEO | RC0402FR -076R8L | 6R8 1% 0402 |
| R3, R4, R8, R11, R25, R26, R33, R34 | 8 | TE | RC0402FR -0718RL | 18R 1% 0402 |

*Continue in the next page*

Table I.3 – *Continued from previous page*

| Designator | Quantity | Manufacturer | Reference | Value |
|---|---|---|---|---|
| R5, R12 | 2 | Panasonic | ERJ-2RKF1003X | 100K 0402 1% |
| R6, R7, R28, R29 | 4 | Yageo | RC0402FR-073R3L | 3R3 0402 1% |
| R13 | 1 | Vishay | CRCW0603 10K0FKEA | 10K 0603 1% |
| R14, R15, R35, R36 | 4 | Panasonic | ERJ-2RKF1302X | 13K 0402 1% |
| R16, R17, R37, R38 | 4 | Vishay | RCWE1206 10L0JTEA | 0R01 1206 1% |
| R18, R19, R27 | 3 | Vishay | CRCW0402 1K00FKED | 1K 0402 1% |
| R22, R23, R24 | 3 | Yageo | RC0402FR-074K02L | 4K02 0402 1% |
| S1 | 1 | Cannon, ITT | KSR231GLFS | |
| V1, V2, V8, V9, V18, V19, V25, V26 | 8 | MCC | 1N4148WX-TP | 0.15A 75V |
| V3, V5, V11, V14, V20, V22, V27, V30 | 8 | Vishay | SISS50DN-T1-GE3 | |
| V4, V6, V12, V13, V21, V23, V28, V29 | 8 | Faircild | RS1M | |
| V10 | 1 | Diodes | B260-13-F | |
| V15 | 1 | ON Semi | FDV305N | |
| X1 | 1 | Amphenol | 95278-801-24LF | |
| X2 | 1 | JST | B02B-PASK-1 | |
| X3, X4, X5 | 3 | Conexcon | BC2EHDRC-02-P | |