This Bachelor's Thesis addresses the design and real implementation of a device capable of carrying out the control of a set of pilots corresponding to several models of high-end vehicles provided by "Valeo® Iluminación", a company based in Martos, Jaén.

The project is approached from a double perspective: on the one hand, the development of the electronic part through the analysis, design and manufacture of a PCB; on the other hand, find out through reverse engineering the control method carried out by the official hardware manufacturer and develop our own.

This wide scope requires applying professional System Engineering methodologies, which minimizes the risk and culminates with the successful completion of the project.

Eric Domek Águila is a Telecommunication engineer from Monachil, Spain. He finished his Bachelor's studies in 2022 at University of Granada, specializing in electronics. He joined GranaSAT team for developing his Bachelor's thesis.

Andrés María Roldán Aranda is the academic head of the present project, and the student's tutor. He is a professor in the Departament of Electronics and Computers Technologies.

BACHELOR'S THESIS

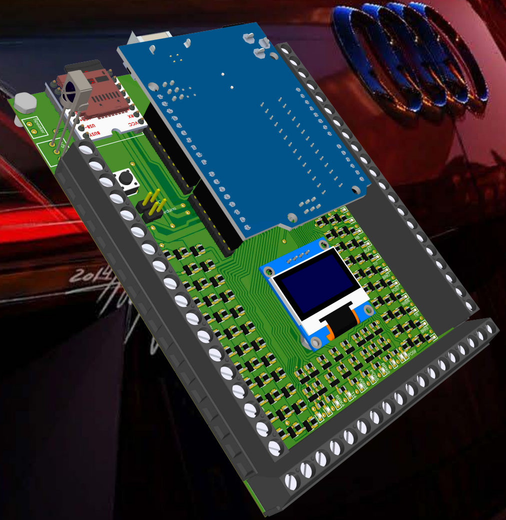
Design of demonstrator device for automotive pilots of high-performance vehicles

Eric Domek Águila

TELECOMMUNICATION ENGINEERING

UNIVERSITY OF GRANADA

Bachelor in Telecommunication Engineering



Design of demonstrator device for automotive pilots of high performance vehicles

Eric Domek Águila

2021/2022

Tutor: Andrés María Roldán Aranda

Credits for the cover: **AUDI**.
Printed in Granada, June 2022.

“Demonstrator device for automotive headlamps/pilots of high-performance vehicles”

DEGREE IN
TELECOMMUNICATION ENGINEERING

Degree Thesis

"Demonstrator device for automotive headlamps/pilots of high-performance vehicles"

ACADEMIC COURSE: 2022

Eric Domek Águila

DEGREE IN TELECOMMUNICATION ENGINEERING

“Demonstrator device for automotive headlamps/pilots of high-performance vehicles”

AUTHOR:

Eric Domek Águila

SUPERVISED BY:

Prof. Andrés Roldán Aranda

DEPARTMENT:

Electronics and Computers Technologies

D. Andrés María Roldán Aranda, Profesor del departamento de Electrónica y Tecnología de los Computadores de la Universidad de Granada, como director del Trabajo Fin de Máster de D. Eric Domek Águila,

Informa:

Que el presente trabajo, titulado:

Dispositivo demostrador para faros/pilotos de automoción de vehículos de alta gama

ha sido realizado y redactado por el mencionado alumno bajo mi dirección, y con esta fecha autorizo a su presentación.

Granada, a 7 de julio de 2022

Andrés Roldan

Fdo. Prof. Andrés María Roldán Aranda

Los abajo firmantes autorizan a que la presente copia de Trabajo Fin de Grado se ubique en la Biblioteca del Centro y/o departamento para ser libremente consultada por las personas que lo deseen.

Granada, a 8 de julio de 2022

[illegible]

[illegible]

Fdo. Eric Domek Águila

Fdo. Prof. Andrés María Roldán Aranda

Dispositivo demostrador para faros/pilotos de automoción de vehículos de alta gama

Eric Domek Águila

KEYWORDS:

Altium Designer® 21, SolidWorks®, Electronic, PCB Design, Automotion, Hardware, Arduino UNO, LIN

ABSTRACT:

In recent years, electronic technology in the automotive world has advanced radically. Practically all the traditional functionalities of a vehicle that were previously implemented with simple mechanical systems, are now controlled by complex electronic circuits and protocols.

One of the aspects that have advanced the most in this regard is that of lighting. The lighting system of a vehicle may seem somewhat banal and insignificant in terms of technological complexity, as it may seem that it is not important enough to give much of itself. However, this is not so. Currently, the headlights of a car are no longer simply a light bulb that turns on or off without showing any kind of intelligence or functionality beyond that, but now, especially in high-end vehicle models, they have a multitude of complex functionalities that provide safety and comfort to the driver. In almost all cases, these functionalities are controlled by complex automotive protocols, which are usually CAN and LIN.

The main purpose of this work is the design and development of a demonstrator device for the operation of a group of automotive pilots/headlights corresponding to high-end vehicles such as the Audi Q5, Audi Q8 and Infiniti Q30.

This objective involves carrying out a reverse engineering process using the "*Exxotest Muxdiag-II*" device and the "*Audi Rearlamps Management*" software on the Audi Q8 2018 headlights controlled by the LIN protocol to extract the frames that make up the communication.

The heart of the project consists of the design from scratch in Altium Designer® 21 of a PCB with its corresponding SCH to which we can connect the set of headlights that we have to carry out its control. In addition, the necessary Firmware for the Hardware to work will also be developed.

In parallel, the manufacture of a wooden support is also carried out in which the set of headlights is comfortably organized to offer a clear visual demonstration of the behavior of the product when it is finished.

This entire development process is carried out at the GranaSAT facilities, an aerospace electronics laboratory at the UGR. At the end of the project development period, the student will not only have managed to pass the "*Bachelor's Thesis*" subject, but will also have gone through the entire process involved in developing a product: becoming aware of the problem to define the requirements, analysis to choose the precise resources, the design of the product itself making important decisions to adapt as much as possible to the initial requirements and, finally, the actual implementation of the product.

Dispositivo demostrador para faros/pilotos de automoción de vehículos de alta gama

Eric Domek Águila

PALABRAS CLAVE:

Altium Designer® 21, Hardware, SolidWorks®, Electrónica, Automoción, Diseño de PCB, SPICE, MATLAB.

RESUMEN:

En los últimos años, la tecnología a nivel electrónico en el mundo de la automoción ha avanzado de manera radical. Prácticamente todas las funcionalidades tradicionales de un vehículo que antes venían implementadas con simples sistemas mecánicos, ahora estan controladas mediante complejos circuitos electrónicos y protocolos.

Uno de los aspectos que más han avanzado en este sentido es el de la iluminación. El sistema de iluminación de un vehículo puede parecer algo banal e insgnificante a nivel de complejidad tecnológica, pues puede parecer que no es lo suficientemente importante como para dar mucho de sí. Sin embargo, esto no es así. Actualmente, los faros de un coche ya no son simplemente una bombilla que se apaga o se enciende sin mostrar ningún tipo de inteligencia o funcionalidades más alla de ello, sino que ahora, sobre todo en modelos de vehículos de alta gama, poseen multitud de complejas funcionalidades que aportan seguridad y comodidad al conductor. En casi la totalidad de los casos, estas funcionalidades son controladas por complejos protocolos de automoción, que suelen ser CAN y LIN.

El proposito principal de este trabajo es el diseño y desarrollo de un dispositivo demostrador del funcionamiento de un grupo de pilotos/faros de automoción correspondientes a vehículos de alta gama como lo son el Audi Q5, Audi Q8 e Inifiniti Q30.

Este objetivo pasa por llevar a cabo un proceso de ingeniería inversa mediante el dispositivo "*Exxotest Muxdiag-II*" y el software "*Audi Rearlamps Management*" a los faros del Audi Q8 2018 controlados por protocolo LIN para extraer las tramas que conforman la comunicación.

El corazón del proyecto consiste en el diseño desde cero en altium de una PCB con sus correspondientes SCH a la que podamos conectar el set de faros del que disponemos para llevar a cabo su control. Además, también se desarrollará el Firmware necesario para que el Hardware funcione.

En paralelo, también se lleva a cabo la fabricación de un soporte de madera en el que cómodamente se organiza el set de faros para ofrecer una clara demostración visual del comportamiento del producto cuando esté terminado.

Todo este proceso de desarrollo se lleva a cabo en las instalaciones de GranaSAT, un laboratorio de electrónica aeroespacial de la UGR. Al final del periodo de desarrollo del proyecto, el alumno no solo habrá conseguido superar la asignatura "*Trabajo de fin de Grado*", sino que habrá pasado por todo el proceso que implica el desarrollo de un producto: tomar conciencia del problema para definir los requisitos, el análisis para elegir los recursos precisos, el propio diseño del producto tomando importantes decisiones para adaptarse lo máximo posible a los requisitos iniciales y, finalmente, la implementación real del producto.

‘Love is the one thing that transcends space and time’

Doctor Brand - Interstellar

Agradecimientos:

Este trabajo ha sido posible gracias a un número muy reducido de personas, pero a las cuales debo un gran agradecimiento; aunque el que pueda mostrarle en estas líneas será sin duda insuficiente, sirva este breve espacio como tal.

El primero de ellos ha de ir destinado a mi familia, mis padres, María y Wojciech, a mi hermana Carolina y a mi abuela Antoñita. Desde siempre, su apoyo incondicional en todos los aspectos ha resultado de vital importancia para que mi motivación permaneciese intacta en todo momento.

A mi tío Paco por ser mi principal fuente de inspiración, modelo a seguir tanto en el ámbito profesional como en el personal y por su insaciable sed de conocimiento. La humanidad aún está por encontrar algo que él no sepa hacer.

A David, Rafa e Iván, que más que amigos ya son hermanos con los que he compartido momentos y situaciones de todo tipo y que siempre han estado ahí para escuchar mis quejas y ayudarme en lo que necesitase.

A Cristian Ruiz, que ha conseguido aguantarme cuando he necesitado consejo técnico durante gran parte del desarrollo del proyecto y sin el cual me habría resultado infinitamente más complicado arrancar.

A todos los compañeros del laboratorio por aguantar meses con la plancha de madera llena de faros de coche molestando por todas partes.

Y por supuesto, a Elena, por acompañarme durante el transcurso de esta larga etapa, y seguir aún sirviendo de apoyo en los peores momentos, por la comprensión y la paciencia, muchas gracias.

Cada uno de los mencionados son partícipes de este Trabajo. A todos ellos, y a los que no pudieron verlo, muchas gracias.

Contents

List of Figures

List of Tables

List of Videos

Glossary

Altium Designer® 21 software used to design PCB from schematics. It allows 3D Design, as well as electronics simulation.

Arduino UNO Electronic board based on Atmel ATmega328 chip.

Black Box is a system which can be viewed in terms of its inputs and outputs (or transfer characteristics), without any knowledge of its internal workings. Its implementation is "opaque" (black)..

Bypass Capacitor is a capacitor that shorts AC signals to ground so that any AC noise that may be present on a DC signal is removed, producing a much cleaner and pure DC signal..

CubeSat Miniaturized satellite normally for space research, with dimensions of $1\,\mathrm{dm}^3$ and mass lower than 1.33 kg per unit.

Datasheet is a document that summarizes the performance and other characteristics of a product in sufficient detail that allows a buyer to understand what the product is and a design engineer to understand the role of the component in the overall system..

EDA Plataforma de herramientas software para el diseño electrónico.

Firmware Is a specific class of computer Software that provides the low-level control for a device's specific Hardware.

Footprint is the arrangement of pads (in SMT) or through-holes (in THT) used to physically attach and electrically connect a component to a PCB..

GranaSAT GranaSAT is an academic project from the University of Granada originally consisting in designing and developing a picosatellite (CubeSat). Coordinated by the Professor Andrés María Roldán Aranda, GranaSAT is a multidisciplinary project with students from different degrees, where they can acquire and enlarge the knowledge necessary to face an actual aerospace project.

Hardware equipment or physical support in computing that refers to the physical, tangible parts of a computer system, its electrical, electronic, electromechanical and mechanical components..

IC Circuit basically composed of semiconductor and passive elements that are manufactured on the surface of a semiconductor wafer and subsequently encapsulated. They make up most of the electronic products on the market.

module In this project we will call a module a set of electronic components that, connected to each other in the correct way, can be seen as a Black Box with a certain number of inputs and outputs whose objective is to perform a specific function within the design..

PAD It is a copper surface on a printed circuit or PCB that allows the component to be soldered or fixed to the board. There are two types of pads; THTs and SMDs.

SCL Arduino UNO I2C clock line.

SDA Arduino UNO I2C data line.

Software Is a collection of instructions and data that tells a computer how to work..

SolidWorks® CAD Software from Dessault Systèmes for 3D Mechanical Design.

Stencil is a sheet of stainless steel with laser-cut openings used to place some solder paste on a PCB board for surface mount component placement..

Acronyms

AC	Altern Current.
ATX	Advanced Technology eXtended.
BCM2	Body Comfort Module.
CAN	Controller Area Network.
CPU	Central Processing Unit.
DC	Direct Current.
ECE	Economic Commission for Europe.
EOBD	European On Board Diagnosis.
I/O	Input/Output.
I2C	Inter-Integrated Circuit.
ID	Identification Designator.
IDE	Integrated Development Environment.
IR	InfraRed.
ISO	International Organization for Standardization.
KWP	Key Word Protocol.
LCD	Liquid Crystal Display.
LDF	LIN Description File.
LED	Light-Emitting Diode.
LIN	Local Interconnect Network.
LSB	Less Significant Bit.
MATLAB	MATrix LABoratory..
MCU	Micro Controller Unit.

MDF	Medium Density Fiberboard.
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor.
MP3	MPEG Audio Layer III.
MPEG	Moving Picture Experts Group.
NCF	Node Configuration File.
OD	Open Drain.
OLED	Organic Light Emmiting Diode.
PC	Personal Computer.
PCB	Printed Circuit Board.
POR	Power Or Reset.
PWM	Pulse Width Modulation.
RAM	Random Access Memory.
ROM	Read Only Memory.
SCH	Schematic.
SD	Secure Digital.
SMD	Surface Mount Devices.
SMT	Surface-Mount Technology.
SPICE	Simulation Program with Integrated Circuit Emphasis.
TFT	Thin Film Transistor.
THT	Through-Hole Technology.
TI	Turn Indicator.
TWI	Two Wire Interface.
UART	Universal Asynchronous Receiver-Transmitter.
UGR	University of Granada.
USB	Universal Serial Bus.
VSP	Virtual Serial Port.
Wi-Fi	Wireless Fidelity.

Chapter 1

Introduction

This bachelor's degree thesis has been developed with the intention of showing a real application of the knowledge acquired during the 4 years of engineering career, as well as during its development, which is of incalculable value. There is no better way to do this than to delve into the development of a real product, which will ensure learning of the engineering process behind the analysis, design and verification of the product. The main objective of this work is to develop a device capable of carrying out a demonstration of the control of a set of pilots from different vehicle models.

At all times during this work, we have tried to accurately follow the actual process of launching a real product. So, during the development of this bachelor's degree thesis, I have not only become familiar with the actual working method for developing a product, but I have also realized how different reality is from theory to reality at time to put my knowledge into practice.

This Bachelor's thesis has been developed in GranaSAT laboratory, which has many multidisciplinary projects that gathers people from a wide variety of fields who are committed to acquiring new knowledge related to Electronics and Aerospace Engineering. Since its origins, one of its main purposes has been getting a CubeSat in orbit ; however, today its goals goes far beyond, and a wide range of devices and projects are developed in collaboration with different students and enterprises.

Furthermore, this bachelor's thesis has been possible thanks to the collaboration with the company "***Valeo** [46] **Iluminacion***", based in Martos, Jaen. The Valeo story started in 1923 in a workshop in Saint-Ouen, just outside of Paris, France. Over 90 years later, it is now a world-leading global automotive supplier operating in 33 countries and partnering with automakers worldwide. This company has provided us with the necessary material to understand and carry out tests on how communication with the drivers is carried out in a real vehicle.

(a) *GranaSAT*

(b) *Valeo logo*

Figure 1.1 – *GranaSAT and **Valeo** logos*

1.1 Motivation

Although since the beginning of time cars already had headlights and from the outside it may seem that the lighting system of a vehicle is something simple and trivial, lighting is one of the fields in which most progress has been made, specially in the last 15 years. From the first oil lanterns to the current lasers and LEDs, almost 150 years have passed in which lighting has evolved a great deal. This progress is logical, not only because of the technological improvements in all these years, but also because lighting is a determining point in active safety (the one aimed at avoiding accidents) in vehicles.

Regarding the evolution of the light source, it could be said that there have been two huge turning points. The first of them was the use of electricity and the second is the development of high brightness LEDs.

In short, the lighting technology used by vehicles has changed drastically in recent years, passing its control from being implemented through the simple ON/OFF of a light bulb to doing it through a switchboard that communicates with the pilots through complex computer protocols.

In addition to showing the complete development process of an electronic product, this work focuses on offering a vision of the technology that is currently used for the control and implementation of the different functions that the headlights of a vehicle can have. This will be done by developing a product capable of controlling the functions of a set of different car headlights with the aim of making possible a demonstration to the public of the operation of the different pilots. In addition to developing the necessary Software for it.

By achieving the project objective, it will be met my own motivation. The motivation that let me develop this project was to acquire knowledge about how the development process of a real functional product is really like. Beside that, I aimed to get the confidence of being able to apply all the theoretical knowledge acquired through my four years of college and obtain the satisfaction of having designed an electronic product from scratch to be able to show its documentation to those interested.

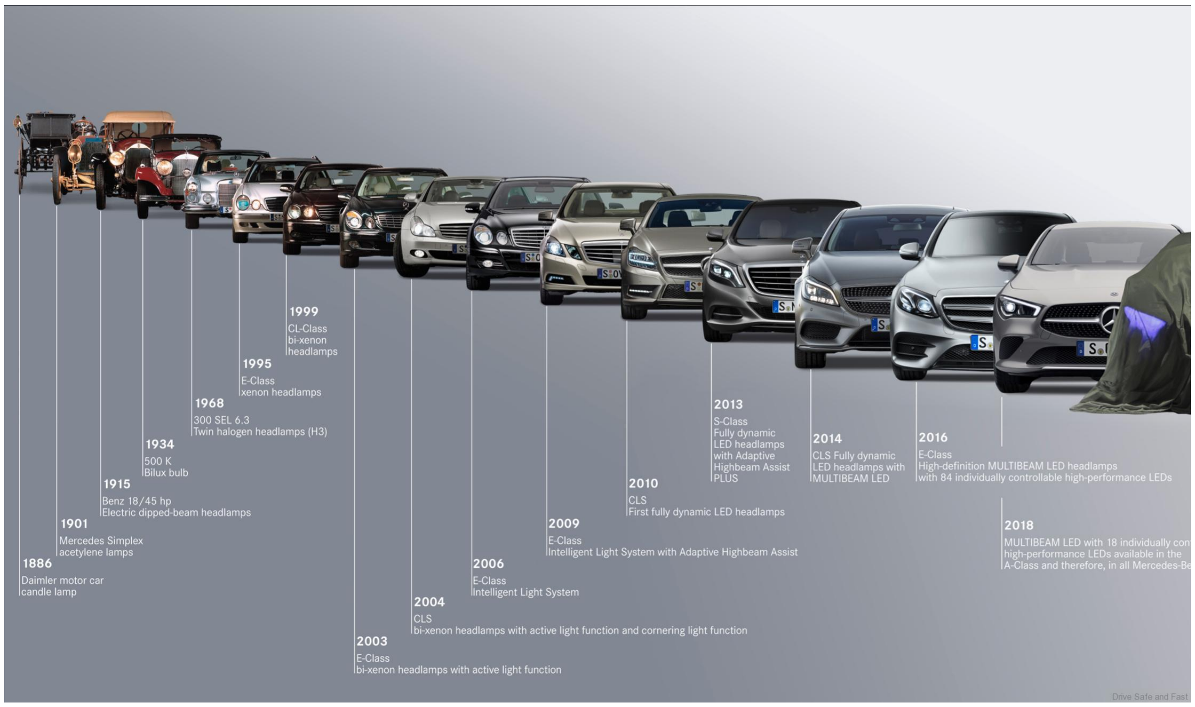

1886
Daimler motor car
candle lamp
1901
Mercedes Simplex
acetylene lamps
1915
Benz 18/45 hp
Electric dipped-beam headlamps
1934
500 K
Bilux bulb
1968
300 SEL 6.3
Twin halogen headlamps (H3)
1995
E-Class
xenon headlamps
1999
CL-Class
bi-xenon
headlamps
2003
E-Class
bi-xenon headlamps with active light function
2004
CLS
bi-xenon headlamps with active light function and cornering light function
2006
E-Class
Intelligent Light System
2009
E-Class
Intelligent Light System with Adaptive Highbeam Assist
2010
CLS
First fully dynamic LED headlamps
2013
S-Class
Fully dynamic
LED headlamps
with Adaptive
Highbeam Assist
PLUS
2014
CLS Fully dynamic
LED headlamps with
MULTIBEAM LED
2016
E-Class
High-definition MULTIBEAM LED headlamps
with 84 individually controllable high-performance LEDs
2018
MULTIBEAM LED with 18 individually con
high-performance LEDs available in the
A-Class and therefore, in all Mercedes-Be
Drive Safe and Fast

Figure 1.2 – *The Evolution of Mercedes-Benz Headlight Tech: Candles to LEDs [4]*

1.2 Project goals and objectives

This section outlines the main top-level non-technical goals of the project. That is, the objectives that are expected to be met at the end of the project and that have nothing to do with the technical specifications of the developed product itself.

Ref.	Formal Requirements
Obj. 1	Become familiar with the LIN protocol by learning the physical specifications of the bus, communication routines, frame format, and usage scenarios.
Obj. 2	Develop a project in Altium Designer® 21 with the complete schematic and PCB of the required device
Obj. 3	Actually manufacture the product
Obj. 4	Acquire a vision of how to work as a team within a laboratory where each one performs a different task according to their professional competence, helping and learning from each other
Obj. 5	Understand the process involved in developing a real product
Obj. 6	To demonstrate the knowledge acquired during the Bachelor's degree in Telecommunication Engineering, as well as multidisciplinary abilities gathered during the execution of this Bachelor's Thesis.
Obj. 7	Pass the subject of "Bachelor's thesis"

Table 1.1 – *Top-level non-technical requirements of the project*

1.3 Project structure

Throughout the development of the project, a multitude of individual tasks have been carried out that have been documented little by little as they were completed. Once the project is finished and when writing this report, a good organization of it is of vital importance, classifying these tasks following the logical order of the development phases of a product.

This project, divided into six chapters and an addendum, progressively analyzes the system under development from different points of view, addresses the design tasks and finalizes with the successful completion of the product. Figure 1.3 shows a diagram that illustrates the structure that has been followed during the project.

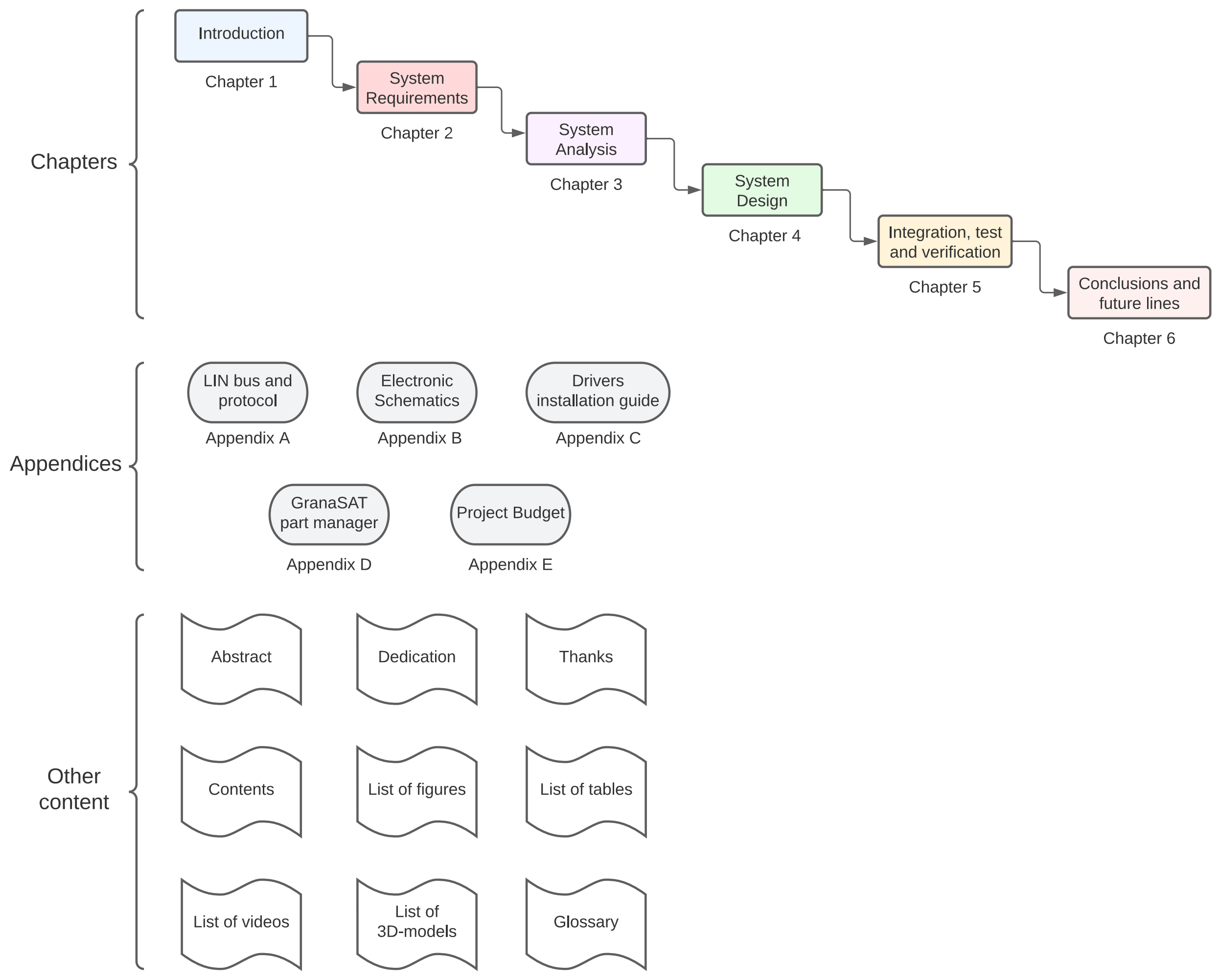


Figure 1.3 – *Project structure made in* `lucidchart.com` *[30]*

The chapters are the following:

- chapter 1: **Introduction**: This chapter provides a very general explanation of what the project to be developed will consist of. It also shows the motivation that justifies the development of the project and the usefulness of the product in the market. High-level non-technical objectives are defined, as well as the structure of the report.
- chapter 2: **System Requirements**: The technical requirements that the product must meet when it is finished are defined. These requirements will be imposed as strict conditions in the analysis and design phases in chapter 3 and chapter 4 respectively.
- chapter 3: **System Analysis**: The modules that must be present in the product design so that it meets all the technical requirements defined in chapter 2 are presented. In addition, the main components that will make up each module are chosen, applying criteria such as price, consumption and the space occupied, but above all the performance required by each module. A Gantt chart is also included that shows the distribution of tasks throughout the period of completion of the work, as well as the economic and power budget.
- chapter 4: **System Design**: From the choices made in chapter 3, we proceed to the schematic and PCB design, defining in depth the particularities of each module, what other components must be connected

and how all the modules are connected to each other for their correct operation and compliance with the technical requirements of the product. In addition, the design of the product's firmware, which makes the hardware do its job, is shown and explained.

- chapter 5: **Integration, test, and verification**: Once the design is finished, this chapter shows the manufacturing process of the same, as well as a series of tests that verify the correct operation of the product.
- chapter 6: **Conclusions and future lines**: Finally, chapter six includes the main conclusions extracted from this Bachelor's Thesis, as well as some future lines of work which have naturally emerged during the design process.
- Appendix A: **LIN bus and protocol**: This appendix shows the characteristics and specifications of the LIN bus, as well as the communication format and particularities of the protocol.
- section B.1: **Electronic Schematics**: Schematic files (*.SCH*) generated with Altium Designer® 21 are shown.
- Appendix C: **Drivers installation guide**: This appendix shows a step-by-step installation guide for the different drivers that have been necessary during the development of the project.
- Appendix D: **GranaSAT part manager**: The use made during the project of the repository of electronic components belonging to the GranaSAT laboratory is shown.

Chapter 2

System requirements

Before starting with the analysis and design of the product, it is necessary to know the **fundamental minimum functions and requirements** that it must meet when it is finished. That is why in this second chapter we will set these formal requirements to establish them as an essential condition during the **system analysis** in chapter 4 and in the **system design** in chapter 5.

It is very important to be clear from the beginning about how we want the product to work when it is finished. In this way, we will be able to keep the main design objectives in mind throughout the project and we will be able to get as close to them as possible with each of the decisions we make.

2.1 Formal Requirements Definition

Since this product will be developed from scratch, we do not have any set minimum performance requirements. However, taking into account all the material we have, we have committed ourselves to the fact that the product **should be capable of carrying out the functions in Table 2.1** from a technical point of view.

To fulfill all these functions we must find and choose the **materials and modules** capable of implementing them. This selection, that will take place in chapter 3, will always be made according to a compromise between **price, fulfillment of the corresponding assigned task and performance.**

It is important to point out that not all these requirements can be met through the practice of a single discipline, so we must be able to identify the knowledge and materials that each one needs (electronic design, component selection, software programming, crafts, etc).

Ref.	Formal Requirements
PR.FoR.1	Being able to **connect and control each function separately of the ON/OFF pilots**.
PR.FoR.2	Being able to **connect and control LIN pilots by connecting it directly to PC**.
PR.FoR.3	Being able to **connect and control LIN pilots without connecting the product anywhere.**
PR.FoR.4	Making **LIN** pilots functions **switch ON and OFF following the rhythm of music**.
PR.FoR.5	Being able to **control the pilots without touching the product**.
PR.FoR.6	Offering a **user-friendly interface** through which you can carry out control of all functions.
PR.FoR.7	Being able to make a **clear demonstration to the public** of the pilots in operation being controlled by the product with the possibility of **transporting all the pilots** in a comfortable and simple way.

Table 2.1 – *Formal Requirements of the product*

Chapter 3

System Analysis

The initial material that we have to carry out this work consists of a set of headlights corresponding to different vehicle models and controlled by different technologies. As we have seen in the Table 2.1, the requirements specify that the product must be able to control each and every one of the functions of each headlight that we have, so the first thing we must take into account before considering what modules are necessary in our product, it is the technology that each headlight needs to respond to the control. Once we know what type of electronic architecture we must implement to meet the control needs of each headlight, we can proceed to the selection of the necessary modules for it.

In addition, our product must also offer the possibility of showing the public a demonstration of the operation of the headlight set in a comfortable and illustrative way. Therefore, an analysis of what would be the best mechanical architecture for it will also be carried out in this section.

As the requirements of the project also specify that it is necessary to develop the Software to control the Hardware that has been designed, it is essential to know how the manufacturer of the headlights controls them when they use them in their vehicles. For this, a reverse engineering process will be carried out through which all the details of the LIN communication will be obtained, which makes possible the operation of the high-end pilots corresponding to the 2018 model of the Audi Q8.

3.1 Headlights technology

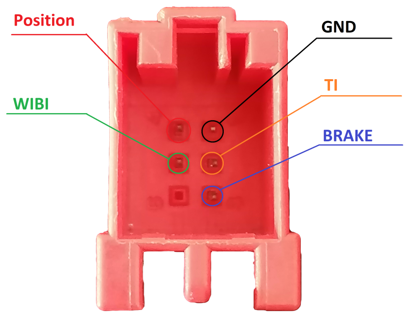
As we have already said, the set of headlights that we have had since the beginning of the project thanks to Valeo consists of headlights corresponding to different vehicle models. In a first approximation, this set of headlights can be divided into two large categories according to the method to control its functions, which is the criterion that most interests us to know which modules are necessary for its control. These categories are:

- **ON/OFF pilots**: Each function available in the headlight is activated separately by feeding the corresponding PIN of its connector, it could be said that the headlight is not intelligent.
- **LIN controlled pilots**: The headlight is powered through a single power PIN (in addition to the ground PIN) and its functions are activated by communicating with the headlight through the LIN protocol, where the pilot is the **slave** and our product would be the **master** (see Appendix A for understanding this topology)

In this section only information corresponding to the ON/OFF type headlights is shown, since all the information related to the headlights controlled by LIN is found in subsubsection 3.4.1.3. We have 3 different types of headlights of the ON/OFF type. Figure 3.1, Figure 3.2 and Figure 3.3 show a front view of each of each of them, its connectors and the corresponding pinout.

(a) *Trunk left light of Renault Megane 2016*

(b) *Trunk right light of Renault Megane 2016*



(c) *Trunk light pinout*

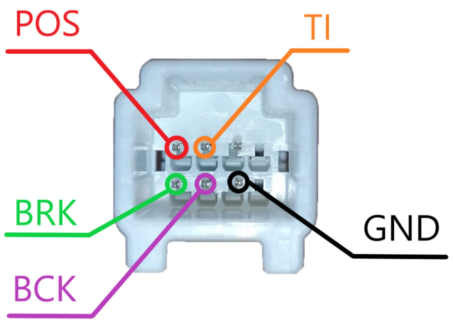
NUMBER	PIN	FUNCTION
1	POS	Position light
2	GND	Ground
3	WIBI	Enables dynamic TI
4	TI	Turn Indicator
5	n.c	n.c.
6	BRK	Brake

(d) *Trunk light pinout description*

Figure 3.1 – *Trunk lights of Renault Megane 2016*

(a) *Trunk LHD Infiniti Q30 2015*

(b) *Trunk RHD Infiniti Q30 2015*



(c) *Pinout*

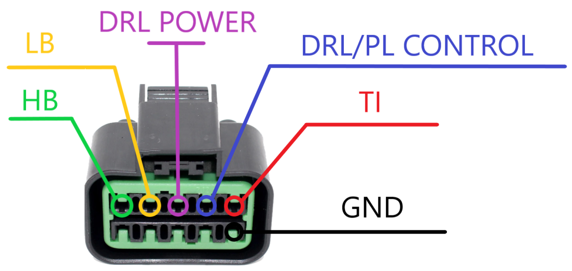
NUMBER	PIN	FUNCTION
1	POS	Position light
2	BRK	Break light
3	TI	Turn Indicator
4	BCK	Back light
5	n.c.	n.c.
6	GND	Ground

(d) *PIN description*

Figure 3.2 – *Trunk lights of Infiniti Q30 2015*

(a) *Right Headlight Renault Megane 2016*

(b) *Left Trunk RHD Infiniti Q30 2015*



(c) *Pinout*

NUMBER	PIN	FUNCTION
1	HB	High Beam
2	LB	Low Beam
3	DRL POWER	Daylight Running Lamps power
4	DRL/PL CONTROL	DRL/Position Lights control
5	TI	Turn Indicator
10	GND	Ground

(d) *PIN description*

Figure 3.3 – *Headlight Renault Megane 2016*

3.2 Electronics architecture

It is clear that most of the required functions specified in the previous section require a **certain electronic architecture** to be carried out. That is why the decision has been made to develop a PCB for this project.

A printed circuit board (PCB) is the board base for physically supporting and wiring the surface-mounted and socketed components in most electronics. Its topology will be explained in more detail in subsection 4.1.2.

In this section we will focus on choosing the different modules that our PCB must have to carry out all the functions required in section 2.1. When making the final decision about which components we will use for the implementation of each of the modules, we will make a comparison taking into account many factors. Although we will always choose the component that best suits the needs of the product with suitable dimensions and price.

3.2.1 Microcontroller/microprocessor module selection

The **microprocessor/microcontroller** is one of the most important modules in the product for the following reasons:

- It will be the **"*brain*" of the product**, since it will be in charge of managing almost everything that has the word "control" in it.
- It will establish important **design conditions** that will influence the choice of the other modules such as:
  - Number of available digital inputs/outputs.

- Processing capacity.
- Number of available serial ports.
- Way of programming it.

Once we have made the choice of the microprocessor we will be able to proceed to the choice of the rest of the components that will be connected to their corresponding pins.

3.2.1.1 Differences between a microprocessor and a microcontroller

The main difference between a microcontroller (ex.: Arduino UNO) and a microprocessor (ex.: Raspberry Pi) is the **input and output** capabilities, as well as the **performance of the CPU.** Depending on the application for which it is desired, it may be better to use one or the other.

The table Table 3.1 shows a comparison of the main common characteristics of microcontrollers and microprocessors.

Microprocessor	**Microcontroller**
Memory and I/O components need to be connected externally	It has a processor along with internal memory and I/O components.
You can't use it in compact systems	You can use it in compact systems.
High cost	Low cost
It's used for general purpose applications that allow you to handle loads of data.	It's used for application-specific systems.
Memory and I/O has to be connected externally, so the circuit becomes large.	Memory and I/O are already present, and the internal circuit is small.
Microprocessor has a smaller number of registers, so more operations are memory-based.	Microcontroller has more register. Hence the programs are easier to write.
It has no RAM, ROM, Input-Output units, timers, and other peripherals on the chip.	It has a CPU along with RAM, ROM, and other peripherals embedded on a single chip.

Table 3.1 – *Differences between microcontroller and microprocessor [28]*

Based on this comparison, we conclude that the best choice is a **microcontroller**, since:

- It will make our system more **compact**, since it has everything necessary in a single microchip.
- It is cheaper.
- Our product will carry out a specific task (the control of the pilots) that will be carried out by the program loaded in the microcontroller.
- The software is easier to write.

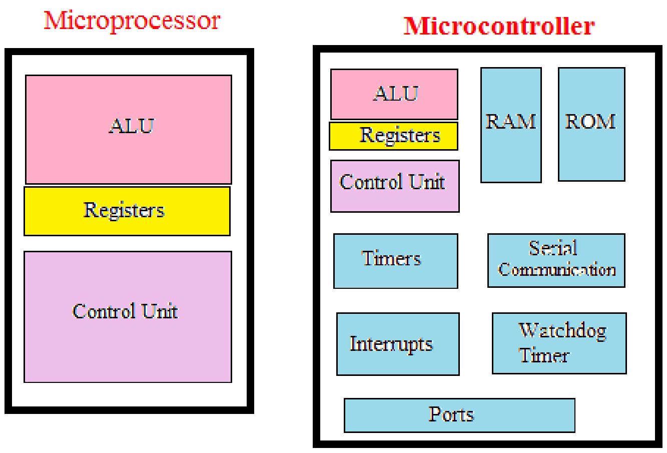


Figure 3.4 – *Microprocessor vs microcontroller internal architecture*

3.2.1.2 Final decision

Next we will make a comparison between 3 different types of microcontrollers and we will choose the one that we consider the best option for our specific application. It should be noted that we are taking into account that all these microcontrollers are mounted on their corresponding Arduino UNO board.

	ATmega328P	**ATmega32u4**	**ATmega2560**
Price	6€ (Aliexpress)	8€ (Aliexpress)	10€ (Aliexpress)
Dimensions	53.4 x 68.6 mm	68.6 x 53.3 mm	101.6 mm x 53.34 mm
Power Supply	7 to 12 V	1.8 to 5.5 V	1.8 to 5.5 V
Current consumption	3.6 to 12.2 mA		
Temperature Range	-40 to 125 ºC	-40 to 85 ºC	-40 to 85 ºC
Flash Memory	32 KB	32 KB	256 KB
Analogic Inputs	6	12	16
Digital I/O	14	20	54
Decision	YES	NO	NO

Table 3.2 – *Microcontroller trade-off*

Taking into account most possible variables, it has finally been decided to use an **ATmega328P** mounted on an **Arduino UNO board**, especially since we already had a free one in the laboratory.

In addition to that, its characteristics are well suited to what we want: it is small, it has a considerable number of digital pins that we can later configure as virtual serial ports and it has more than enough flash memory for the program that is required.

It should be noted that it is possible to purchase the microcontroller itself separately instead of purchasing it **mounted on a development board** like the one shown in the Figure 3.5, thus taking up less space. However, we have decided to **use the entire development board**, since it is **easier to remove and put on our PCB** and, in addition, there is software that offers a **more friendly programming and monitoring environment: Arduino IDE**.

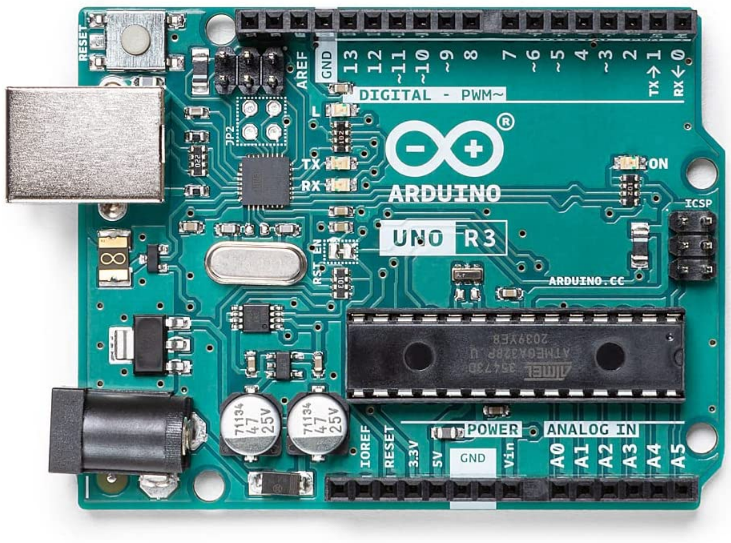


Figure 3.5 – *Arduino UNO board [1]*

3.2.2 Power supply module selection

The way in which we will **supply power to the circuit** is another of the most important decisions we have to make when selecting the components that make up the different modules of our design. That is because each specific component will need to be powered with a **certain voltage** and will consume a **certain current**, which will make it necessary:

- To have access on the board to the **different voltage levels** required by the modules.

- That the power supply used is capable of **supplying the necessary power** to power all the modules of the design.

It stands to reason that we need to know these details first before we can make a decision on how to feed our design. In the following sections of this chapter, this data corresponding to both the candidate components to be chosen and those finally selected to be part of the project will be shown. In addition, both in Appendix E and in section E.1, a summary of the power budget is shown, where the power consumed by each component and the total power required by the design can be easily and quickly observed.

Attending to the power requirements seen in the appendices mentioned, we proceed to choose the way in which we will power our board. To do this, first of all, we must choose what type of energy source we will use, several options quickly occur to us:

- **Using a battery**: A battery is a device consisting of two or more electrochemical cells that can **convert stored chemical energy into electrical current**. Each cell consists of a positive electrode, or cathode, a negative electrode, or anode, and electrolytes that allow ions to move between the electrodes, allowing current to flow out of the battery to carry out its function: powering a electric circuit. Table 3.3 shows the comparison between pros and cons of using a battery to power our product. Figure 3.6 shows how an actual lead-acid battery used in vehicles looks like.

PROS	CONS
Full portabilty of the product, not needing any wires	Limited life
Standard item, easy to find	It would increase the weight of the product
	It would be necessary to recharge it or change it every time
	We would need a very high capacity one for our application

Table 3.3 – *PROS and CONS of using a battery to supply our product*

Figure 3.6 – *Lead-acid battery*

- **Using a linear power supply**: These sources are designed to offer a **low noise level**. They are used in applications requiring **low electromagnetic emissions and good transient response.** Normally they convert the AC voltage coming from the electrical network into one (or several) much smaller DC voltages. Linear power supplies can only step down an input voltage to produce a lower output voltage.

PROS	CONS
Can work for as long as needed	It is usually large and heavy
Offers different voltage levels at the output	Reduces the portability of our product
Provides great power	Not very efficient

Table 3.4 – *PROS and CONS of using a linear regulated power supply to power our product*

Table 3.4 shows a comparison of the pros and cons of using a regulated linear power supply to power our product. Figure 3.7 briefly shows in a block diagram the inner workings of a linear regulated power supply.

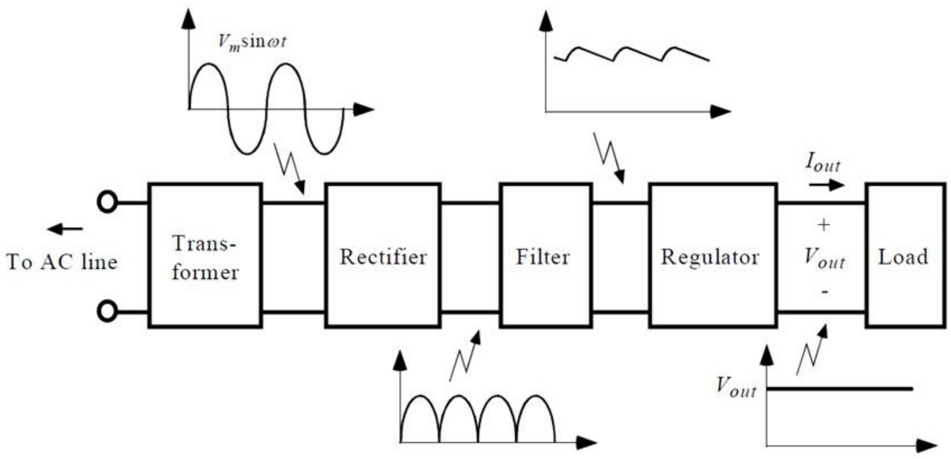


Figure 3.7 – *Linear regulated power source internal block diagram [38]*

- **Using a switching power supply**: They are designed to achieve great efficiency and take up as little space as possible. They use PWM modulation to regulate the output voltage efficiently.

Figure 3.8 – *ATX switching power supply*

Table 3.5 offers a quick overview of the pros and cons of using a switch mode power supply to power our product. This type of sources is usually used for applications that require a **high current supply.**

PROS	CONS
Very high efficiency	Limits the portability of the product
Offers different voltage levels at the output	Produces high-frequency noise
Provides great power	
They are small in size	
They are usually cheap	

Table 3.5 – *PROS and CONS of using a switching power supply to power our product*

Finally, taking into account all the positive and negative facts regarding the different options that have been considered to power our product, it has been decided to use a **switching power supply.** In general, this option is the one that best suits our needs, since, although it **greatly limits the portability of the product**, this is not critical, since, normally, we will **not need to move it while it is in use**. In addition, high-frequency noise is not a problem either, since the product **does not work at these frequencies.** Its small size, great efficiency and low price have been the determining engineering criteria to make this decision.

Specifically, the source that we will use will be an ATX, a switching power supply belonging to an old PC already available in the laboratory, so it will not be necessary to purchase a new one.

3.2.3 LIN Transceiver module selection

To meet the requirements:

- "***PR.FoR.2. Being able to connect and control LIN pilots by connecting it directly to PC.***"
- "***PR.FoR.3. Being able to connect and control LIN pilots without connecting the product anywhere.***"

specified in section 2.1 it is very important to note that neither the USB port of the PC nor the Arduino UNO itself offer **interfaces for LIN communication**. That is why we will need to include a **LIN transceiver module in our design**.

It will be an essential component, as it will be responsible for adapting the two-wire I2C serial communication to a one-wire LIN communication.

As we have done with the choice of microcontroller in subsection 3.2.1, we have decided to make a comparison between **3 models** from different integrated circuit manufacturers that implement the LIN transceiver function. The data is displayed in the Table 3.6

	ATA6662C	**MCP2003/4**	**NCV7327**
Manufacturer	Atmel	Microchip	Mouser
Price	0.65 € (Aliexpress)	1.18 € (Aliexpress)	1.04 € (Mouser)
Package	SOP-8	SOP-8	SOP-8
Power Supply	5 V to 27 V	6 to 27 V	5 to 18 V
Current Consumption	Sleep: 10-20 uA Normal: 1.2-2 mA Fail safe: 0.5-1.1 mA	90-150 uA	Sleep: 8-10 uA Normal (rec.): 0.2-1.2 mA Normal (dom.): 2-6.5 mA
Temperature Range	-55 to 150 ºC	-40 to 125 ºC	-40 to 150 ºC
Transmission Rate	Up to 20 KBaud	Up to 20 KBaud	Up to 20 kbps
Decision	Yes	No	No

Table 3.6 – *LIN Transceiver trade-off*

Finally, we opted for the **ATA6662C from Atmel**. Especially because of the **difference in price**, since, as we can see, almost all basic LIN transceivers have similar characteristics. In addition, for our specific application, characteristics such as consumption are not critical, which is where our choice mainly loses out.

3.2.4 USB port module selection

As we have specified in the section 2.1. One of the essential requirements is: "***PR.FoR.2. Being able to connect and control LIN pilots by connecting it directly to PC***".

We think that the most sensible solution to establish a connection between our product and the computer is a **USB port**. Currently there are different types of USB ports, in this section we will focus on choosing the most suitable for the requirements demanded by the desired application of our product.

	Type-A	**Type-B**	**Type-C**
STANDARD	USB 1.1 USB 2.0 USB 3.2	USB 1.1 USB 2.0 USB 3.2	USB 1.1 USB 2.0 USB 3.2 USB 4
TRANSFER SPEED	12 Mbps to 10 Gbps	12 Mbps to 10 Gbps	12 Mbps to 40 Gbps
DIMENSION	13.10 x 17.70 x 11.75 (mm3)	Type B: 12.04 x 7.78 x 21.60 Mini B: 3 x 7 x 1.45 (mm3)	8.94 x 8.37 x 3.14 (mm3)
PRICE	2.73 €	Type B: 2.20 € Micro B: 0.594 €	2.74 €
ELECTION	Yes	No	No

Table 3.7 – *USB port trade-off*

For the application that concerns us, we are not limited by the maximum transfer speeds of any of the three types of USB showed in Table 3.7. Taking this into account, we will select the most suitable USB type depending mostly on **their size and price**. Since we do not need a super fast connection, we will not take that feature so much into account.

We will choose the **USB micro-B** in light of the fact that it is the **most affordable and smallest connector.**

3.2.5 USB to Serial module selection

When we carry out serial communications through the TX and RX pins of the Arduino, we are indirectly using a converter present in it that is responsible for adapting USB communication to **serial communication**.

Attending to the same requirement as in subsection 3.2.4: "***PR.FoR.2. Being able to connect and control LIN pilots by connecting it directly to PC***"; we will need a device capable of converting the **USB communication** present in the selected *Mini USB-B* port to ***RS-232* serial communication (TX and RX).**

In the Arduino Datasheet, we can see that this microcontroller mounts an **CH340** to carry out the task that we want in this section. Searching on the internet, it **is possible to purchase the CH340 separately**, what's more, we have not found actual alternatives in the market to this to carry out this task, so we will focus our choice between the **different variants of it**.

	CH340-G	**CH340-C**	**CH340-T**
MANUFACTURER	WCH	WCH	WCH
VOLTAGE RANGE	Min.: 4.5 V Typ.: 5 V Max.: 5.3 V	Min.: 4.5 V Typ.: 5 V Max.: 5.3 V	Min.: 4.5 V Typ.: 5 V Max.: 5.3 V
CURRENT CONSUMPTION RANGE	Typ.: 12 mA Max.: 30 mA	Typ.: 12 mA Max.: 30 mA	Typ.: 12 mA Max.: 30 mA
PACKAGE	SSOP-16	SSOP-16	SSOP-20
XTAL	NO	YES	NO
PRICE	0.38 € (es.aliexpress.com)	0.5 € (es.aliexpress.com)	0.69 € (es.aliexpress.com)
ELECTION	No	Yes	No

Table 3.8 – *USB to serial trade-off*

Finally, as we can see in the Table 3.8, we have chosen the **CH340-C**, since it has the **package with the fewest pins** of all the models (16). In addition, the most important thing is that the **CH340-C** has the **crystal oscillator (XTAL)** necessary for the correct operation of the serial communication inside, so it is not necessary to include one externally, which **saves us using a component, space and money.**

3.2.6 Shift Register module selection

In accordance with the formal requirement: " ***PR.FoR.1. Being able to connect and control each function separately of the ON/OFF pilots***" present in section 2.1, we will need a sufficient number of digital outputs to be able to control each function of the ON/OFF lights separately.

As we have specified in subsubsection 3.2.1.2, the **ATmega328P** chip only has **14 digital outputs**. For our case we will need a total of **32**, so we have to find a solution.

The most viable solution that has occurred to us has been to make use of **shift registers** with **serial-in** and **parallel-out**. This kind of **shift registers** are devices capable of generating a **parallel digital output** of a certain number of bits from a **single serial input**.

	SN74HC595	**74ALS164**	**CD4094**
MANUFACTURER	ON-Semiconductor	Philips	Texas Instruments
VOLTAGE RANGE	Min.: 2 V Typ.: 5 V Max.: 6 V	Min.: 4.5 V Typ.: 5 V Max.: 5.5 V	Min.: 3 V Typ.: 5 V Max.: 18 V
CURRENT CONSUMPTION	1 uA	10 mA	1 uA
BITS OUTPUT	8	8	8
PACKAGE	TSSOP-16	SO-14	SOP-16
TEMPERATURE RANGE	-55 to 125 ºC	-65 to 150 ºC	-55 to 125 ºC
CASCADE	YES	YES	YES
PRICE	0.09 € (es.aliexpress.com)	2.22 € (es.aliexpress.com)	0.16 € (es.aliexpress.com)
ELECTION	YES	NO	NO

Table 3.9 – *Shift register trade-off*

As we can see in Table 3.9, all three options we have chosen as candidates for our **shift register module** have an **8-bit parallel output**. This is because in the market it is almost impossible to find shift registers with a higher number of output bits, among other things, because **it is not practical** because most common uses can be easily implemented by chaining together several smaller shift registers.

This leads us to think that the main characteristic that will determine our final choice will be the **ability to connect several of these shift registers in cascade**, thus forming a shift register with the output bits that we need.

Reviewing the Datasheets of each of the 3 candidates, in one way or another, they all implement the possibility of making this connection in cascade, so the absence of this feature does not rule out any for now.

Taking this into account, the main factor that leads us to choose the **SN74HC595** is the price and the extremelly low power consumption, in addition, it has a more than sufficient **temperature range**.

3.2.7 Audio player module selection

According to the requirement "***PR.FoR.4. Making LIN pilots functions switch ON and OFF following the rhythm of music.***" specified in section 2.1. We will need some module capable of playing music from the audio samples generated by the microcontroller chosen in subsubsection 3.2.1.2.

Having based our choice of microcontroller on the use of an Arduino UNO board, we have the advantage of looking for our MP3 player among the multitude of existing accessories created especially for use in conjunction with Arduino development boards, which will make us the both software and hardware connection work much easier.

As we can see in the Table 3.10, this time we have opted for the **Arduino DF Player Mini**. One of the reasons is that it is **smaller size**. But the determining factor in the final decision was its **much lower price**.

	JQ8400	**DF Player Mini**
MANUFACTURER	Chinese	Arduino
DIMENSIONS	18 x 25 mm	20 x 20 mm
VOLTAGE RANGE	3.3 - 5.2 V	3.2 - 5 V
CURRENT	Normal: 10 mA Sleep: 0.5 mA	Min.: 20 mA Max.: 150 mA
TEMPERATURE RANGE	-40 to 85 ºC	-40 to 70 ºC
PRICE	2.27 (es.aliexpress.com)	0.97 € (es.aliexpress.com)
ELECTION	NO	YES

Table 3.10 – *MP3 player trade-off*

Is important to note that the **JQ8400** is more efficient in terms of **power consumption**, plus it has a **sleep mode** where it consumes a minimal amount of power when not in use. However, choosing the **DF Player Mini** due to its lower price will not make a significant difference in the performance of our final product.

This MP3 player supports playback of files either from a microSD card or from the **TX and RX** pins of the microcontroller's **serial communication**. For our application we will use the second method.

3.2.8 Power amplifier module selection

Attending to the requirement "***PR.FoR.1. Being able to connect and control each function separately of the ON/OFF pilots***". It is essential to realize that the current that, in general, the functions of the ON/OFF pilots need is **much greater** than the maximum current that the digital outputs of the Arduino UNO are capable of providing and, therefore, the outputs of the shift registers. This maximum current is **20 mA** and the pilots need currents of the order of hundreds of mA.

In addition, let us remember that the **voltage** necessary to activate the functions of the pilots is **+ 12 V DC**, being the voltage provided by the digital pins of the Arduino UNO of **+ 5 V DC**.

To solve this problem we have thought of implementing a power amplifier module. For this we will need a **MOSFET transistor**. This transistor must be **P-Channel** for the reason explained in [].

	IRLML6402TRPBF [37]	**FDN342P** [40]	**FDN360P** [41]
MANUFACTURER	Infineon	Fairchild semiconductor	Fairchild semiconductor
PACKAGE	SOT-23	SOT-23	SOT-23
D-S MAX. DC VOLTAGE	20 V	20 V	30 V
MAX. DC CURRENT	3.7 A	2 A	2 A
TEMPERATURE RANGE	-55 to 150 ºC	-55 to 150 ºC	-55 to 150 ºC
PRICE	0.03 € (es.aliexpress.com)	0.02 € (es.aliexpress.com)	0.09 € (es.aliexpress.com)
ELECTION	NO	YES	NO

Table 3.11 – *MOSFET for power amplifier module trade-off*

As can be seen in Table 3.11, three candidates are proposed to be the transistor that will form the power amplifier module that we need. This time, we have decided to opt for the **FDN342P**, since it is the most economical option which meets the requirements that we've been able to find.

3.2.9 Remote control module selection

We have multiple options to meet the requirement "***PR.FoR.5. Being able to control the pilots without touching the product***". Quickly, 3 options come to mind.

3.2.9.1 Wi-Fi module

Using a Wi-Fi module: There are several devices capable of implementing a Wi-Fi connection working together with our microcontroller Arduino UNO.

PROS	CONS
It would be possible to control the device from any location in the world as long as internet access is available.	The connection is more complex
We could control our product from various types of terminals: smartphones, laptops, PCs, etc.	We would expose our product to possible computer attacks
Apart from controlling it, we could access more information about the device.	There may be problems if the Wi-Fi connection is not stable
	Are usually expensive
	Take up a lot of space on our PCB

Table 3.12 – *PROS and CONS of using a Wi-Fi module to control our product*

3.2.9.2 Bluetooth module

Using a ®Bluetooth module: There are several devices capable of implementing a ®Bluetooth connection working together with our microcontroller Arduino UNO.

PROS	CONS
It's a fast connection	The connection is complex
We could control our product from various types of terminals: smartphones, laptops, PCs, etc.	The connection has a limited range
Apart from controlling it, we could access more information about the device.	It can lose connection in certain conditions
No line of sight hence can connect through any obstacles.	Security is a very key aspect as it can be hacked
	It has low bandwidth as compared to Wi-Fi

Table 3.13 – *PROS and CONS of using a Bluetooth module to control our product*

3.2.9.3 IR receiver module

Using an IR receiver: It would be necessary to use an IR remote control to send signals to the IR receiver, which would receive them and send them to a specific digital PIN of the Arduino UNO, which would process them and act accordingly.

PROS	CONS
It provides a very secure communication	It requires sender and receiver to have direct vision
Very low power consumption	It can only control the device, not receiving information of it
They are physically smaller in size	
They are very low-cost (even we have one free in the lab)	
Very fast response time	

Table 3.14 – *PROS and CONS of using an IR receiver to control our product*

As we can see in the Table 3.12, Table 3.13 and Table 3.14 the pros and cons of each possible option to meet the specified requirement have been evaluated. In green, the most **relevant positive factors** are represented when making the decision; the most **relevant negative factors** are represented in red, and the **less determining factors**, but also existing ones, are represented in orange.

3.2.9.4 Final decision

Finally we have decided to make use of an **IR receiver**. Since it is clear that it has the largest number of highly relevant positive factors and the less number of highly relevant negative factors. In short: **it's cheap, small, fast and will do the job.**

In this case, there are no relevant options on the market with which to compare the IR receiver made for Arduino UNO, the **VS1838** so we will go for that one. Table 4.11 shows the main characteristics of VS1838.

3.2.10 Display module selection

According to the requirement "***6. Offering a user-friendly interface through which you can carry out control of all functions.***" specified in section 2.1, we have thought of different solutions:

3.2.10.1 7 segment display

A **seven-segment display** is a form of electronic display device for displaying decimal numerals or letters. Seven-segment displays are widely used in digital clocks, electronic meters, basic calculators, and other electronic devices.

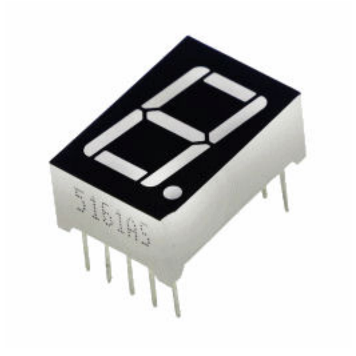


Figure 3.9 – *7-segment display*

Table 3.15 shows a comparison of the pros and cons of using an LCD screen as a display module.

PROS	CONS
Extremely cheap	Zero possibility of working graphically
Minimal risk of malfunction	We would have to use several of them
Easy to program	Only monochrome
	Poor efficiency

Table 3.15 – *PROS and CONS of using a 7-segment display as display module*

3.2.10.2 LCD display

An LCD display is a thin, flat screen made up of a number of color or monochrome pixels placed in front of a light source or reflector. It is often used in battery-powered electronic devices as it uses **very small amounts of power**.

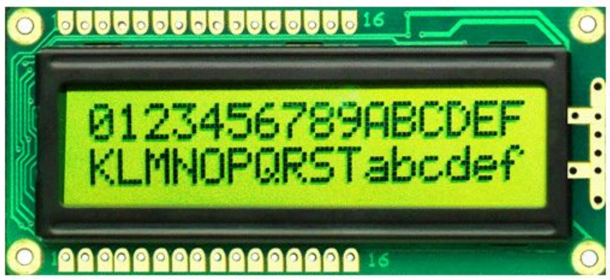


Figure 3.10 – *LCD display*

Table 3.17 shows a comparison of the pros and cons of using an LCD screen as a display module.

PROS	CONS
Low power consumption	Limited Viewing Angles
Minimal Heat Production	Little-to-No Refresh Rate Flicker
Lots of code available online for programming	Is extremely difficult to work graphically
	Expensive

Table 3.16 – *PROS and CONS of using a LCD display as display module*

3.2.10.3 OLED display

OLEDs are one of the types of screen available to use in our electronics and Arduino UNO projects.

An OLED is a type of LED in which the emissive layer is made up of an organic compound that emits light in response to electricity.

Like all other types of screens, OLEDs need a specific controller that converts the received data into electronic signals to control the screen.

PROS	CONS
Possibility of graphical working	Big space occupied
Large field of view	
Lots of code available online for programming	
Relatively cheap	
Relatively power efficient	

Table 3.17 – *PROS and CONS of using an OLED display as display module*

In Figure 3.11 we can see an example of an **SSD1306** OLED display for Arduino UNO.

Figure 3.11 – *OLED display*

3.2.10.4 Final decision

Taking into account the pros and cons of the different options, we have decided to use an **OLED display**, for the reason that we need the ability to **work graphically** with the display to program a **friendly and intuitive user interface**. In addition, the **difference in price and energy efficiency is not critical** as is shown in Table 3.19. Finally, there are very small OLED screens that will **avoid the problem of space.**

	7-Segment	LCD	OLED
PRICE RANGE	0.78 to 1€	3 to 3.5 €	1.78 to 2 €
CURRENT CONSUMPTION RANGE	10 to 70 mA aprox.	2.5 mA aprox.	18 mA aprox.

Table 3.18 – *Price and power consumption comparison between 7-segment, LCD and OLED displays*

The Table 3.18 shows the comparison between 5 possible displays. 3 of them are OLED and 2 LCD have also been included in order to show their characteristics compared to OLED, although we already know that the choice will be between one of the OLED ones.

	LCD + SD 8TFT 128x160	**SSD1331 OLED**	**SSD 1306 OLED** [44]	**ILI9341** [24]	**TFT LCD module HT0280CI01BR1** [23]
SUPPLY VOLTAGE	-0.3 to 4.6 V	2.4 to 3.5 V	1.65 to 3.3 V	2.5 to 3.3 V	2.5 to 3.3 V
CURRENT		Sleep Mode: 10 uA Max Supply Current: 500 uA	Sleep Mode: 10 uA Max Supply Current: 150 uA	80 mA	80 mA
TEMPERATURE RANGE	-30 to 85 ºC	-40 to 85 ºC	-40 to 85 ºC	-20 to 70 ºC	-20 to 70 ºC
RESOLUTION	128 x 160 1.8"	96 x 64 0.95"	128 x 64 0.96"	240 x 160 2.8"	240 x 320 2.8"
COLORS	65 thousand RGB format	65 thousand	Monochrome	RGB Format	RGB Format
TOUCH PANEL	NO	NO	NO	YES	YES
DIMENSION	58 x 34 mm^2	30.7 x 27.3 x 11.3 mm^3	26.7 x 19.26 x 1.85 mm^3	56 x 35 x 1.41 mm^3	50 x 69.2 x 3.45 mm^3
PRICE	$21 www.hotmcu.com	5.20 € www.hotmcu.com	7.99 € www.hotmcu.com	12.76 € www.hotmcu.com	7.25 € www.hotmcu.com
ELECTION	NO	NO	YES	NO	NO

Table 3.19 – *Display module trade-off*

On this occasion we have opted for the **SSD1306**, since, for a very **low price**, we have a very small OLED screen, which is quite favorable to the criterion of occupying the mini space of our PCB. In addition, it is very **energy efficient** and is capable of working in a **wide range of temperatures**.

3.2.11 Terminal connectors module selection

Finally, we will need to implement some physical solution to be able to connect all the drivers to our board. To carry out this task, we have decided to use terminal blocks like the one in Figure 3.12.

These are connectors that are used to connect **bare cables** (without any type of male connector) **directly to the board**, since they have a **screw** to adjust the connection and prevent them from coming loose. These terminal blocks can withstand a maximum current of **13.5 A**.

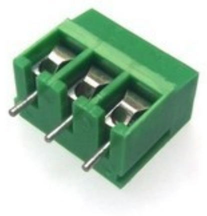


Figure 3.12 – *Terminal Block [36]*

3.3 Mechanical architecture

3.3.1 Demonstrator wooden plank

As we have said in section 2.1, one of the requirements of our product is "***PR.FoR.7. Being able to make a clear demonstration to the public of the headlights in operation being controlled by the product with the possibility of transporting all the pilots in a comfortable and simple way.***"

To meet this requirement, it has been decided to use a **wooden plank** on which all the available pilots will be attached so that they can **all be seen** at the same time in a fancy way, in turn, in a **fully functional manner**, that is, every function of each pilot can be **accessed and connected easily** once assembled.

3.3.1.1 Attachment method

To carry out this task, it is logical to first look at the way in which each pilot is originally designed to be attached to the corresponding vehicle model. Especially since, if we find that the pilots do not have a feasible anchoring method to attach them to a wooden plank, we will have to discard this option and think of another. Once we know that, we can think of some feasible method to be able to organize the pilots in some way that meets the requirements.

Looking at each of the headlights, we realize that some of them have male threads on the back to join the car and other ones have holes. It has been decided to use these threads and holes as methods of anchoring the headlights to the wood.

3.3.1.2 Measurements

First of all, we must **measure the metric and the pitch** of the screw threads present in the different headlights that we have seen in the previous section. In this way we will be able to know the **measurements of the other components** that we will have to use to anchor the headlights.

To carry out this measurement, a **pitch gauge** has been used. A tool capable of measuring the **pitch and metric** of the thread of a screw.

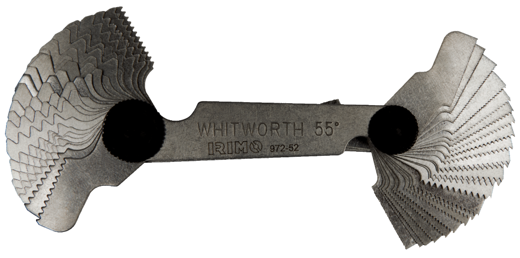


Figure 3.13 – *Pitch gauge*

The measurements obtained are shown in Table 3.20.

	Trunk (long)	Fender (left/right)	Trunk lamp (LH/RH)	Forward lamp	Trunk LHD
Metric	M6	No screws	M5	No screws	M6
Pitch [1] (mm)	1	No screws	0.8	No screws	1
Number	8	No screws	3	No screws	3

Table 3.20 – *Headlights screws pitch and metric measurements*

Next we will measure the dimensions of the headlights themselves to have an idea of what they would all occupy once we distribute them on the wood.

	Trunk (long)	Fender (left/right)	Trunk lamp (LH/RH)	Forward lamp	Trunk LHD
Length (cm)	152	44	61	73	26
Width (cm)	10	17	22	40	12
Depth (cm)	24	21	16	15	11

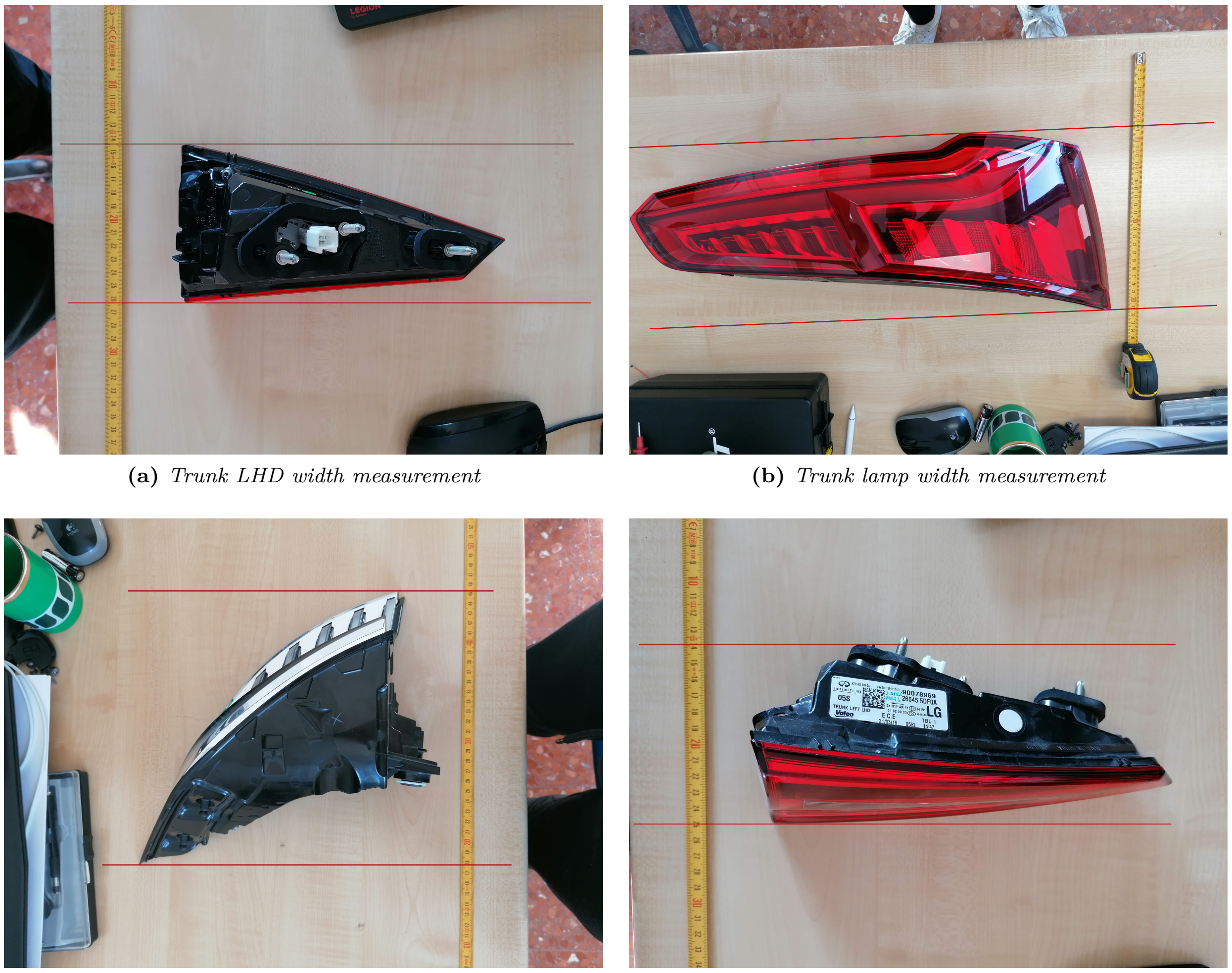
Table 3.21 – *Headlights dimensions measurements*

As we only want to get an idea of the **approximate dimensions of the headlights**, we do not need millimeter precision for their measurement. That is why the method we have used has been to photograph the headlight next to a tape measure. Subsequently, using ***"Photoshop"*** software, we have drawn parallel straight lines starting from the ends that we want to measure and cutting with the tape measure. In this way, subtracting the extreme values we obtain the desired measure.

In Figure 3.14 we can see some examples where the aforementioned measurement method is applied. We have not included a photograph of all the measurements because basically the method is the same and it would interfere with the reading.

Once we have all the measurements corresponding to both the dimensions of the headlights and the pitch and metric screws, we are ready to decide on the components that we must acquire to complete the anchoring of all the headlights to the wood. The design of how to do this is explained in subsection 4.2.1.

[1]Pitch: pitch measurements refer to the distance the screw moves in or out when given one turn. In general, for each metric value, there are two types of pitch: normal and fine.


(a) *Trunk LHD width measurement* **(b)** *Trunk lamp width measurement*

(c) *Fender left depth measurement* **(d)** *Trunk left depth measurement*

Figure 3.14 – *Some headlights measurement method example*

3.4 Software reverse engineering

In order to subsequently design the necessary software to establish communication with the LIN pilots through the designed hardware, we first need to know the exact details of the LIN communication that the pilots obey in order to function. To do this, we will carry out a reverse engineering process called **"bus sniffing"** through which we will obtain all the frames present in the communication.

3.4.1 Material used for the obtention of the LIN schedule

Once the operation and characteristics of the LIN bus have been explained and are clear in Appendix A, we proceed to explain the procedure followed in the laboratory to extract the frames corresponding to the LIN communication established between the PC and the LIN pilots. For this we begin by showing and explaining the material that has been necessary for it:

- Software *"AUDI Rearlamps Management v1.1.13"* 3.4.1.1 provided by Valeo.
- *"Exxotest USB-MUXDIAG-II"* device 3.4.1.2 provided by Valeo.

- Set of three Audi Q8 LIN-controlled pilots provided by Valeo.

- Logic Analyzer 24 MHz, 8 channels. 3.4.1.4

3.4.1.1 Software *"AUDI Rearlamps Management"*

In Figure 3.15 we can see the interface of the *"AUDI Rearlamps Management v1.1.13"* Software once selected the Q8 model. This interface is mainly divided into two parts:

- **Interactive area** for the elaboration of the data frame that will be sent to the set of pilots controlled by LIN to activate the corresponding function.

- **Passive diagnosis zone** of the different functionalities of the pilots according to the data received from them. In addition, this area is divided into two parts depending on the position of the pilots in the vehicle (left or right)

For the purpose of this work we will focus our attention only on the interactive part of frame generation, since we are not going to intentionally activate errors in the pilots to capture the data received in the passive diagnostic zone.

The interactive part of the software basically allows us two ways to configure the command that will be sent to the pilots and that will turn on the desired function:

- Manually, using the switches on the left side of the interface that modify the BCM2 message bit by bit.

- With preset functions that correspond to certain configurations of the switches mentioned above, using the buttons on the right side of the interface

Once the connection of the pilot(s) has been made, we can press the green "start generations" button. If the pilots have been connected correctly, their corresponding lights on the gray panel located just to the right of the button will light up green, which will mean that the connection between the PC and the pilot(s) has been successfully made and frames are already being sent through the *“Exxotest USB-MUXDIAG-II”* 3.4.1.2.

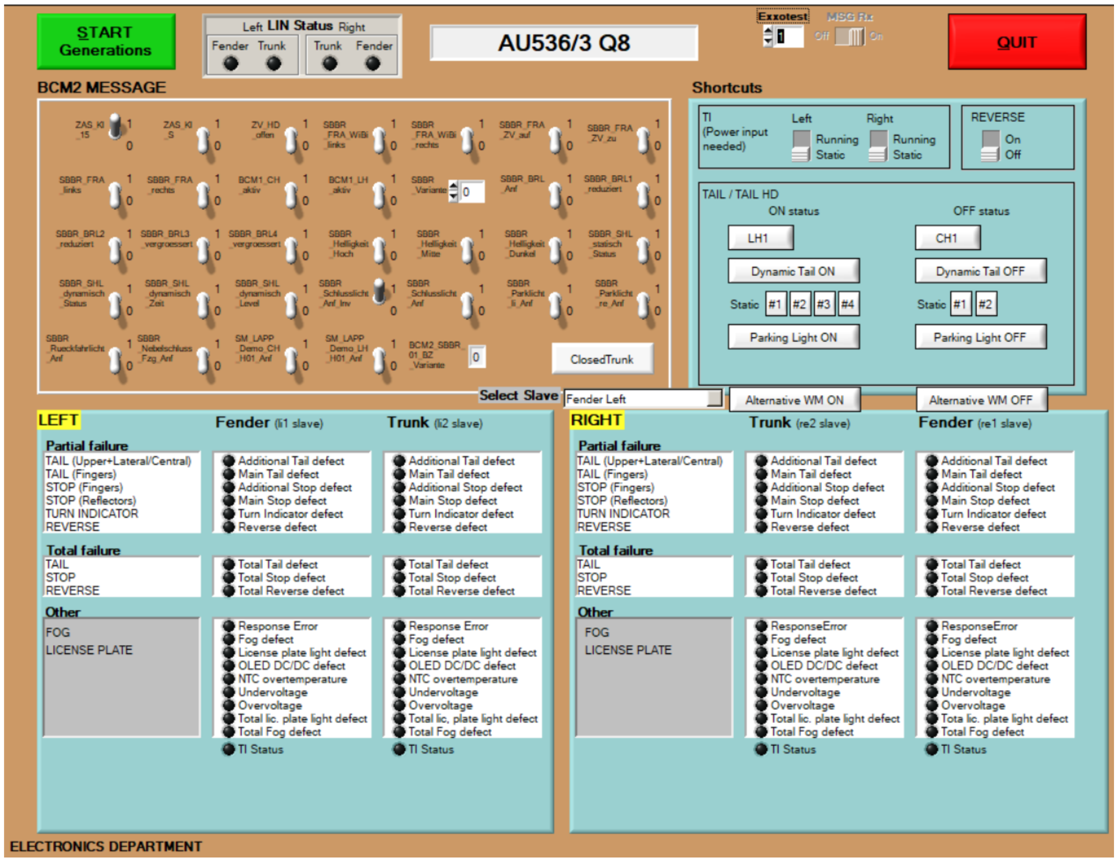


Figure 3.15 – *Audi Rearlamp Management v1.1.13 interface*

3.4.1.2 Exxotest USB-MUXDIAG-II

In this section we will see all the information related to the Exxotest USB-MUXDIAG-II device and how to use it to establish a LIN communication with the pilots.

Figure 3.16 – *Exxotest USB-MUXDIAG-II*

The Exxotest USB-MUXDIAG-II allows to interface a PC (or a pocket PC) with the CAN and KWP2000[2]

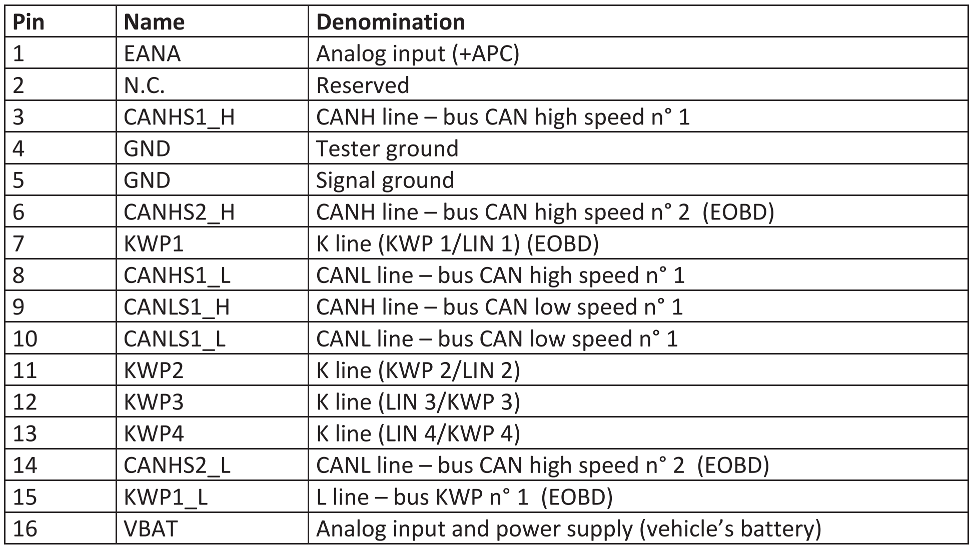
[2]KWP2000: is an acronym for Key Word Protocol 2000, an application layer communications protocol used in automotive electronics and standardized by the International Organization for Standardization with the ISO 14230 code. It is basically

diagnostic channels of a vehicle using an USB link. The interface has the following channels:

- 1 CAN high speed or CAN low speed – fault tolerant channel to be chosen through the software.
- 1 CAN high speed channel (Norme ISO 11898)
- 2 LIN channels master or slave or ISO9141 to be chosen through the software.
- 2 ISO9141 channels or LIN master to be chosen through the software.
- 2 analog inputs (1 is used for the power supply survey)
- 100 μ sec clock for events timing

The diagnostic channels are managed by the KWP2000 protocol (ISO14230) for K line communication, or by the DiagOnCAN protocol (ISO15765) for CAN communication. The USB-MUXDIAG-II interface is powered by the USB port of the linked computer or by the linked vehicle's battery.

The Figure 3.17 shows the pinout of the J1962 connector that Exxotest USB-MUXDIAG-II mounts.



Pin	Name	Denomination
1	EANA	Analog input (+APC)
2	N.C.	Reserved
3	CANHS1_H	CANH line – bus CAN high speed n° 1
4	GND	Tester ground
5	GND	Signal ground
6	CANHS2_H	CANH line – bus CAN high speed n° 2 (EOBD)
7	KWP1	K line (KWP 1/LIN 1) (EOBD)
8	CANHS1_L	CANL line – bus CAN high speed n° 1
9	CANLS1_H	CANH line – bus CAN low speed n° 1
10	CANLS1_L	CANL line – bus CAN low speed n° 1
11	KWP2	K line (KWP 2/LIN 2)
12	KWP3	K line (LIN 3/KWP 3)
13	KWP4	K line (LIN 4/KWP 4)
14	CANHS2_L	CANL line – bus CAN high speed n° 2 (EOBD)
15	KWP1_L	L line – bus KWP n° 1 (EOBD)
16	VBAT	Analog input and power supply (vehicle's battery)

Figure 3.17 – *J1962 Exxotest USB-MUXDIAG-II connector pinout*

In Figure 3.18 we can see a synoptic that summarizes quite well in a block diagram the internal distribution of the different modules of the Exxotest USB-MUXDIAG-II.

used for off-board diagnosis of electronic control units of vehicles

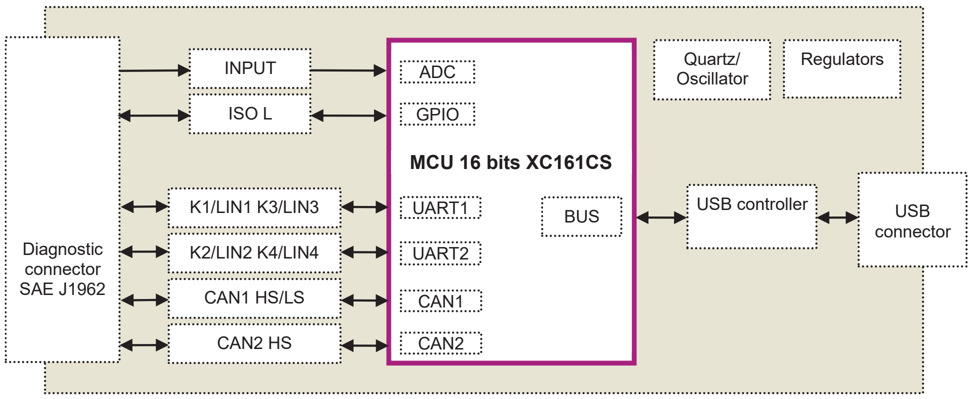


Figure 3.18 – *Exxotest USB-MUXDIAG-II synoptic*

As we can see on Figure 3.18, using a USB controller (similar to a CH340) the conversion from USB to UART [3] is done within Exxotest USB-MUXDIAG-II itself.

There are 2 LIN buses available on the Exxotest USB-MUXDIAG-II, both connected to a separate UART of an MCU.

For our task, we will only need one of them, since, as we have seen in the section where the LIN protocol is explained, multiple slave devices (each of our 4 LIN pilots) are controllable with a single LIN bus or cluster.

In addition, we will need the power pins (VBAT and GND) that will mark the high and low levels of the bits that make up the frames through which the protocol is communicated.

In summary, of all the 16 pins that the J1962 connector in Figure 3.17 has we will be interested in for our task in the ones in Table 3.22

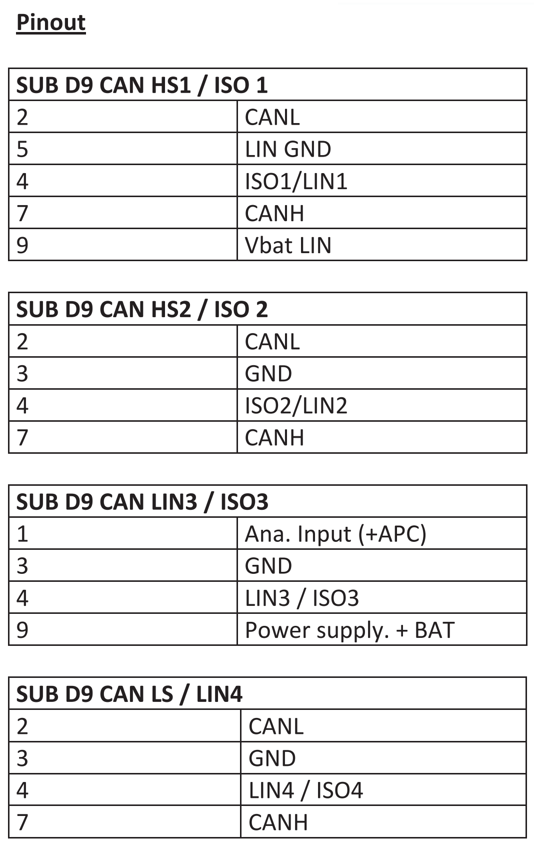
NUMBER	PIN	FUNCTION
7	KWP1	K line (KWP 1/LIN 1) (EOBD)
5	GND	Signal ground
16	VBAT	Analog input and power supply

Table 3.22 – *Necessary J1962 PINs for LIN bus sniffing*

For greater convenience when accessing the 3 necessary pins present in the J1962 connector in Figure 3.17, we will use an adapter called **AMUX-2C2L** shown in Figure 3.19 (a).

[3]UART: a device that controls the serial ports

(a) *Fender left light of Audi Q8 2018*



Pinout

SUB D9 CAN HS1 / ISO 1	
2	CANL
5	LIN GND
4	ISO1/LIN1
7	CANH
9	Vbat LIN

SUB D9 CAN HS2 / ISO 2	
2	CANL
3	GND
4	ISO2/LIN2
7	CANH

SUB D9 CAN LIN3 / ISO3	
1	Ana. Input (+APC)
3	GND
4	LIN3 / ISO3
9	Power supply. + BAT

SUB D9 CAN LS / LIN4	
2	CANL
3	GND
4	LIN4 / ISO4
7	CANH

(b) *Fender right light of Audi Q8 2018*

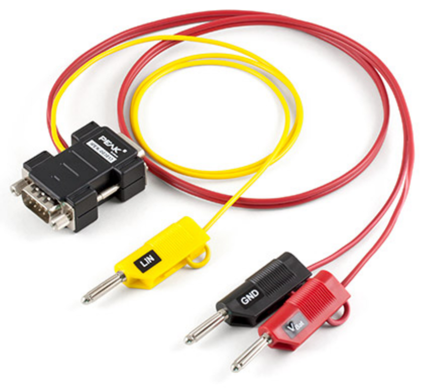
Figure 3.19 – *J1962 to DB9 adapter AMUX-2C2L*

This adapter will allow us to translate the J1962 connector pins to 4 DB9 connectors as we can see on Figure 3.19 (b). To access the 3 pins we need, just use the first of the 4 DB9 connectors (SUB D9 CAN HS1 / ISO1), which will give us the PINs in Table 3.23.

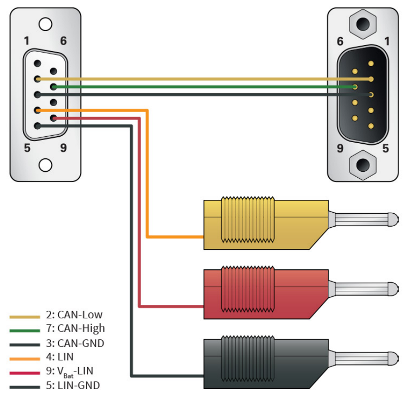
PIN	Function
5	Signal GND
4	LIN
9	VBAT LIN

Table 3.23 – *3 PINs needed for LIN communication*

Finally, it would be a good idea to provide independent access (without using any connectors) to the three pins we need. To convert the male DB9 connector to 3 independent pins, we will use the adapter in Figure 3.20 (a) whose pinout is shown in Figure 3.20 (b), which consists of a male DB9 connector from which the 3 pins that we need separately come out.



(a) *Fender left light of Audi Q8 2018*



(b) *Fender right light of Audi Q8 2018*

Figure 3.20 – *DB9 to 3-pin adapter*

The “*Exxotest USB-MUXDIAG-II*” drivers installation tutorial has been moved to section C.1 in order to no interfere with the reading.

3.4.1.3 Complete LIN pilot set

Thanks to the company ***“Valeo iluminacion”*** [46] based in Martos (Jaén), we have the complete set of 3 rear lights of a 2018 model of an Audi Q8 for the realization of this end-of-degree project.

Figure 3.21 – *Audi Q8 rear*

These are **high-end pilots**, so their functionalities are controlled by the LIN protocol, unlike the other lights we have, whose functions are controlled by turning their corresponding power ON/OFF. In the set of LIN pilots we have:

- 1 trunk light: It has a long LED bar that joins the two side lights of the trunk.
  - Trunk left
  - Trunk right

Figure 3.22 – *Trunk light of Audi Q8 2018*

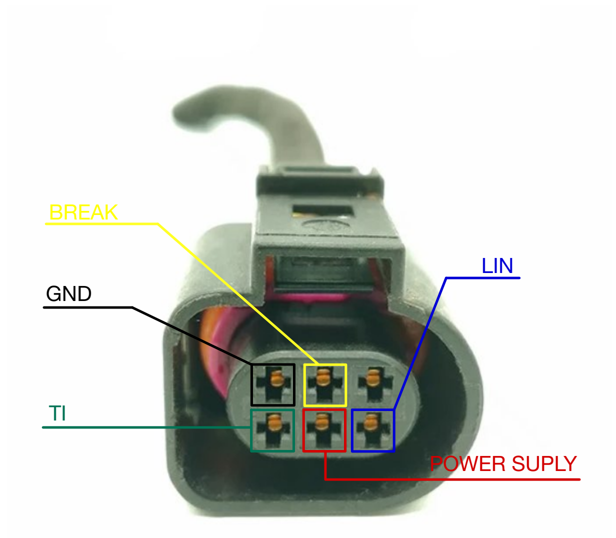
- 2 "fender" lights
  - Fender left
  - Fender right

(a) *Fender left light of Audi Q8 2018*

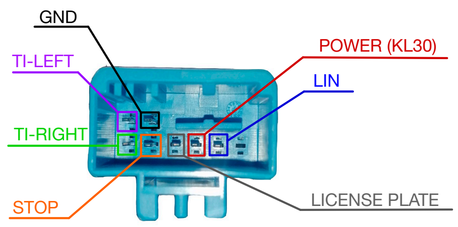
(b) *Fender right light of Audi Q8 2018*

Figure 3.23 – *Fender lights of Audi Q8 2018*

Figure 3.24 shows the type of connector used by each pilot to connect the car. In addition, a diagram will be made where the functions corresponding to each PIN of the connector will be specified.



(a) *Fender left/right female connector 1J0.973.713*



(b) *Trunk left and right male connector*

Figure 3.24 – *Audi Q8 LIN pilots connectors pinouts*

Depending on the pilot model, there are different pinout variants for the fender lights connector, which are shown in Table 3.24

	ECE Basic	**ECE High**	**SAE Basic**	**SAE High**
PIN 1	Supply (KL30)	Ground	Supply (KL30)	Supply (KL30)
PIN 2	Brake	Brake	Ground	TI/Brake
PIN 3	TI	TI	n.c.	Ground
PIN 4	Ground	Supply (KL30)	TI/Brake	n.c.
PIN 5	LIN	LIN	LIN	LIN

Table 3.24 – *Trunk Left/Right Pinout Types*

In our case, we have the **ECE High** model, so we will refer to the corresponding pinout. The Figure 3.24 (a) shows the type of connector used by these two pilots, called **1J0.973.713**. We can also see the function that each pin has associated with our specific pilot model.

We can notice that, although all the functions of the pilot can be controlled through the LIN protocol, feeding the pilot in a general way through the power pin, there are also **independent pins** that we can power separately to **activate each function independently** in the event of a fault in the LIN communication system.

To control the two trunk lights, there is really only one connector through which the long LED bar is also controlled. Figure 3.24 (b) shows the trunk light connector shape and it's corresponding pinout, having the same possibility of controlling the different functions separately, as we have mentioned.

3.4.1.4 Logic Analyzer

Once we have installed the Exxotest USB-MUXDIAG-II 3.4.1.2 device drivers as explained in section C.1 and made the connection showed in subsection 3.4.2 correctly, we are ready to start the LIN bus sniffing. For this we will use a different device called "**logic analyzer**".

A **Logic Analyzer** is an electronic instrument that captures and displays multiple signals from a digital system or digital circuit.

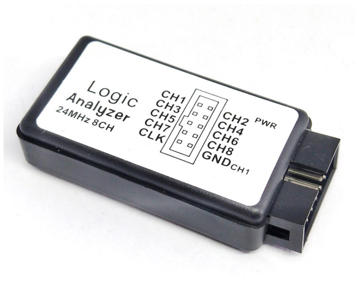


Figure 3.25 – *Logic Analyzer, 24 MHz, 8 channels*

It may convert the captured data into **timing diagrams**, state machine traces, assembly language, or may correlate assembly with source-level software.

Logic analyzers have advanced triggering capabilities, and are useful when a user needs to see the timing relationships between many signals in a digital system.

In any case, the most interesting function of the logic analyzer to carry out the LIN bus sniffing is the **protocol decoding**, which will allow us to obtain the LIN frames that circulate through the bus with relative ease. Below is the Datasheet of the logic analyzer, where its characteristics are shown in detail [39].

saleae
www.saleae.com

Saleae Logic 8 USB Logic Analyzer

FEATURES

- Powerful, Easy-to-use Software
- Deep Sample Buffers
- Highly Portable, USB Attached
- 24 Included Protocol Analyzers
- Automation API
- Custom Protocol Decoder Plugin API
- Edge and Pulse Width Triggering
- Protocol Result Filter and Search
- Measurements, Bookmarks and Timing Markers
- Four Data Export Formats: CSV, Binary, VCD and MATLAB
- Cross Platform Windows, Linux, and OSX

APPLICATIONS

- Firmware Debugging
- FPGA Debugging
- Functional Verification
- Performance Profiling
- Reverse Engineering
- Protocol Decoding
- Data Logging

KEY SPECIFICATIONS

- Eight Digital Channels
- 100 MSPS Digital Sampling (max)
- 25 MHz Max Digital Bandwidth
- Eight Analog Channels
- 10 MSPS Analog Sampling (max)
- 1 MHz Analog Bandwidth
- Recording Length Limited by Available RAM and Density of Recorded Data
- RGB LED, Customizable 24 bit Color

DESCRIPTION

The Saleae Logic 8 USB Logic Analyzer is an 8 channel logic analyzer with each input dual purposed for analog data recording. The device connects to a PC over USB and uses the Saleae Logic Software to record and view digital and analog signals.

A logic analyzer is a debugging tool used to record and view digital signals. It operates by sampling a digital input connected to a device under test (DUT) at a high sample rate. These samples are recorded to a sample buffer, and at the end of the capture, the buffer is displayed in the software for review.

Logic analyzers are great for debugging embedded applications. In the most common case, a developer working on firmware for a microcontroller will write code to communicate with another component, possibly using protocols like serial, I2C, or SPI. To verify the functionality or to diagnose errors in the firmware, a logic analyzer is connected to the digital IO used for communication and records the activity during testing. The recording is then shown on the display so the user can view the actual behavior of the firmware, and compare that with the expected behavior to narrow down and identity the source of the issue – or verify that the operation is correct.

INCLUDED COMPONENTS

Saleae Logic 8 USB Logic Analyzer, 2x 4 Channel Wire Harnesses, 16 Micro-Gripper Hooks, Saleae Carrying Case, USB micro cable, and a Getting Started Guide

PIN CONFIGURATION

Channel 0 [Digital + Analog]	Channel 1 [Digital + Analog]	Channel 2 [Digital + Analog]	Channel 3 [Digital + Analog]
Ground	Ground	Ground	Ground

Channel 4 [Digital + Analog]	Channel 5 [Digital + Analog]	Channel 6 [Digital + Analog]	Channel 7 [Digital + Analog]
Ground	Ground	Ground	Ground

ABSOLUTE MAXIMUM RATINGS

Input Voltage	-25.00V to +25.00V
Operating Temperature	0°C to +70°C

OPERATING RATINGS

Input Voltage	+0.00V to +5.00V
Temperature	0°C to +70°C

ELECTRICAL CHARACTERISTICS

Input Impedance	1 MΩ || 10 pF
Digital Sampling Rates	100***, 50***, 40***, 25, 20, 10, 8, 5, 4, 2, 1 MSPS
Analog Sample Rates	10, 5, 2, 5, 1.25 MSPS, 625, 125*, 5*, 1* KSPS, 100*, 10* SPS
Digital Logic Threshold	V_{IL} +0.6V, V_{IH} +1.2V
Common Supported Logic Standards	+5.0V, +3.3V, +2.5V, +1.8V, RS-232, RS-485/RS-422, +12V
Digital Bandwidth	25 MHz
Analog Bandwidth (-3db)	1 MHz**
ADC Number of Bits	10
Analog Input Voltage Range	+0.0V to +5.0V
Analog Volts per LSB	4.88 mV
PC Connection	USB 2.0 High Speed

Notes:
*Planned sample rates.
**Bandwidth when sampling at 10 MSPS.
***100 MSPS digital on up to 3 channels, 50 MSPS on up to 6 channels, 40 MSPS on up to 7 channels

3.4.1.5 Software used

In order to "translate" the information captured by the logic analyzer, we will need to use software that can interpret it. This software is the own of the company "Logic": ***"Logic 2.3.39"***.

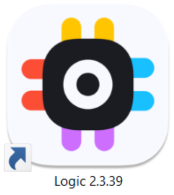


Figure 3.26 – *Logic 2.3.39 logo*

When connecting the logic analyzer via USB to the PC, it will automatically install the corresponding drivers to recognize it. In the same way, “Logic 2.3.39” will recognize it as a logic analyzer ready to receive data on the channels that are connected.

In Figure 3.27 we can see an example of using the logic analyzer software, which is doing the sniffing of 4 channels: 2 digital and 2 analog.

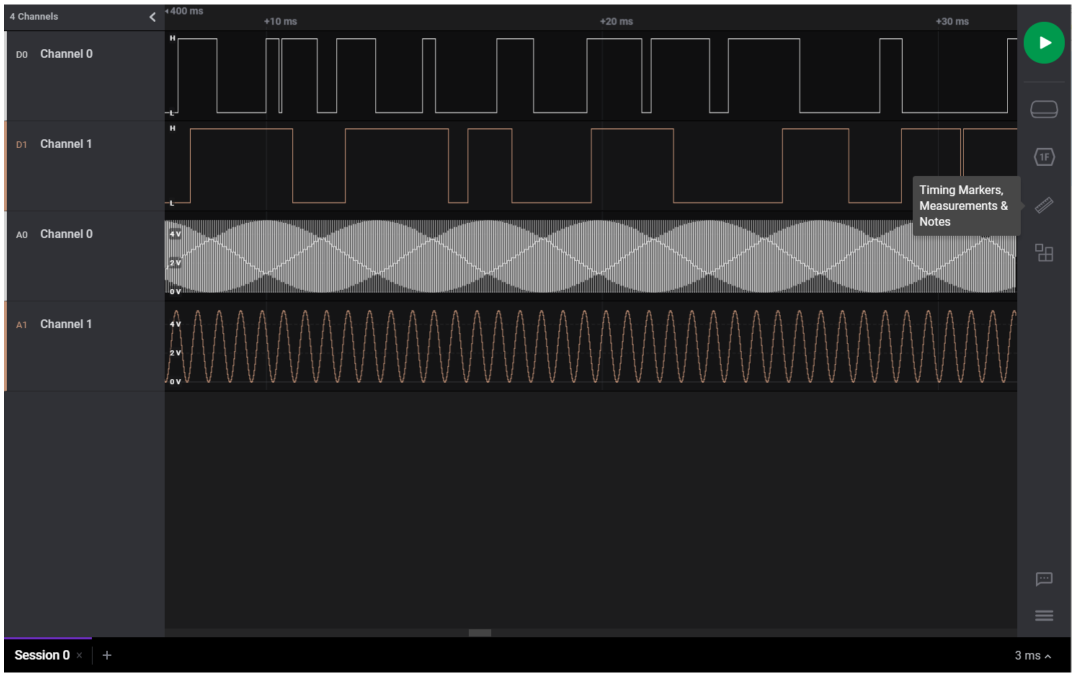
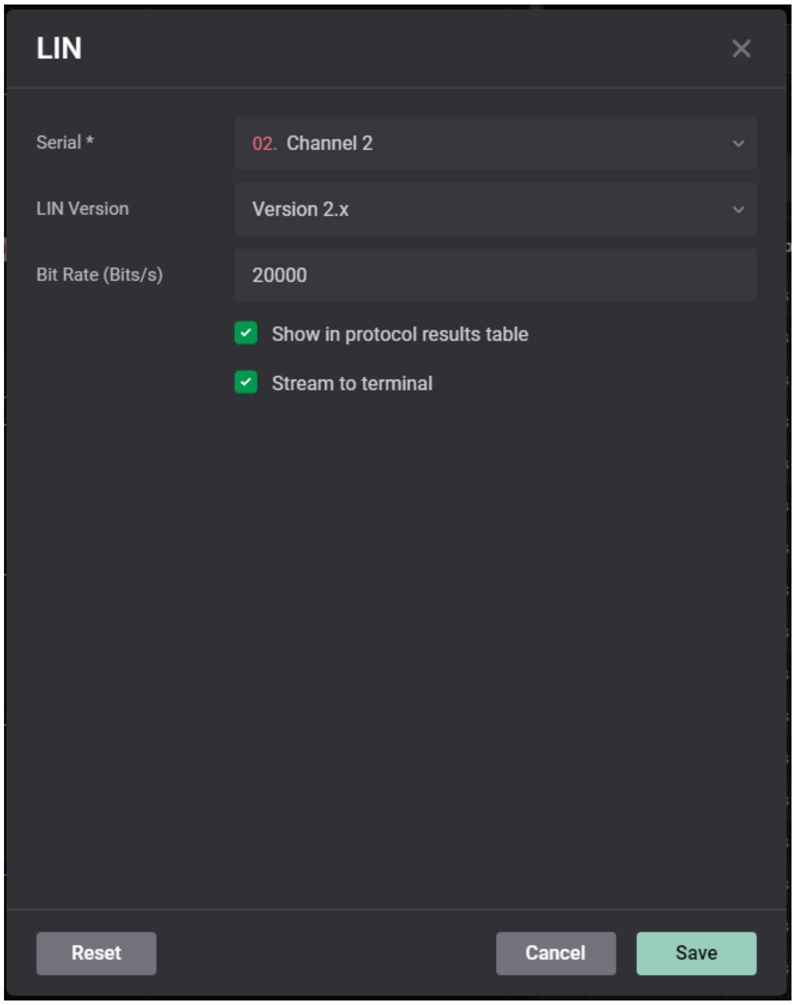


Figure 3.27 – *Logic Analyzer sniffing example*

In the configuration of each channel, it is possible to select the specific protocol that is going to be read in it. In this way, the software saves us the process of interpreting the frames in hexadecimal or binary format.

The Figure 3.28 shows how the program must be configured so that it "knows" that a LIN communication is taking place on channel 2 (in this case) and can show us the different fields of the decoded frame. First of all, we must select the **channel** in which the communication will take place. We can also specify the version of the LIN protocol that is used, as well as the communication **speed**. Finally, there is the option for the software to show us the results of the decoding in a **comfortable table** (or in the terminal).


LIN

Serial * 02. Channel 2

LIN Version Version 2.x

Bit Rate (Bits/s) 20000

✓ Show in protocol results table

✓ Stream to terminal

Figure 3.28 – *Configuration for LIN decoding using Logic 2.3.39*

The Figure 3.29 shows a real example of how the software interprets/decodes the bits to decode the corresponding messages (a LIN frame header in this case) within the LIN protocol.

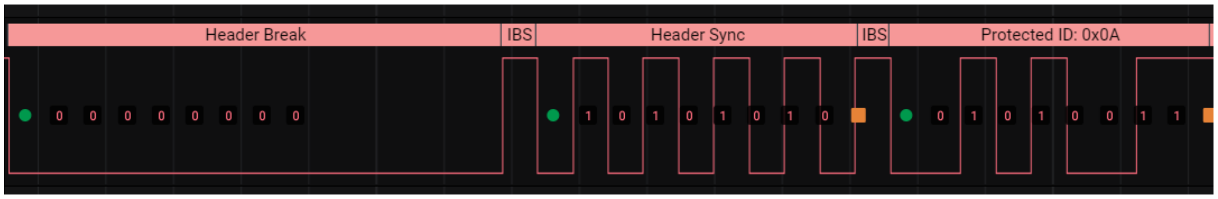


Figure 3.29 – *Example of decoding the header of a LIN frame using Logic 2.3.39*

3.4.2 Connection diagram

In order to be able to obtain in the software the frames that circulate through the LIN bus, we can connect any channel of the logic analyzer, for example, channel 1, directly to the LIN bus through the resistor named **R1**. The value of this resistance is not critical, but its existence is, since it serves to create a potential difference between the logical values 0 and 1 of which the communication consists. If this resistor is not included, the analyzer will not be able to read any frames.

To perform the sniffing, **R1 = 1kOhm** has been used, which has given us good results.

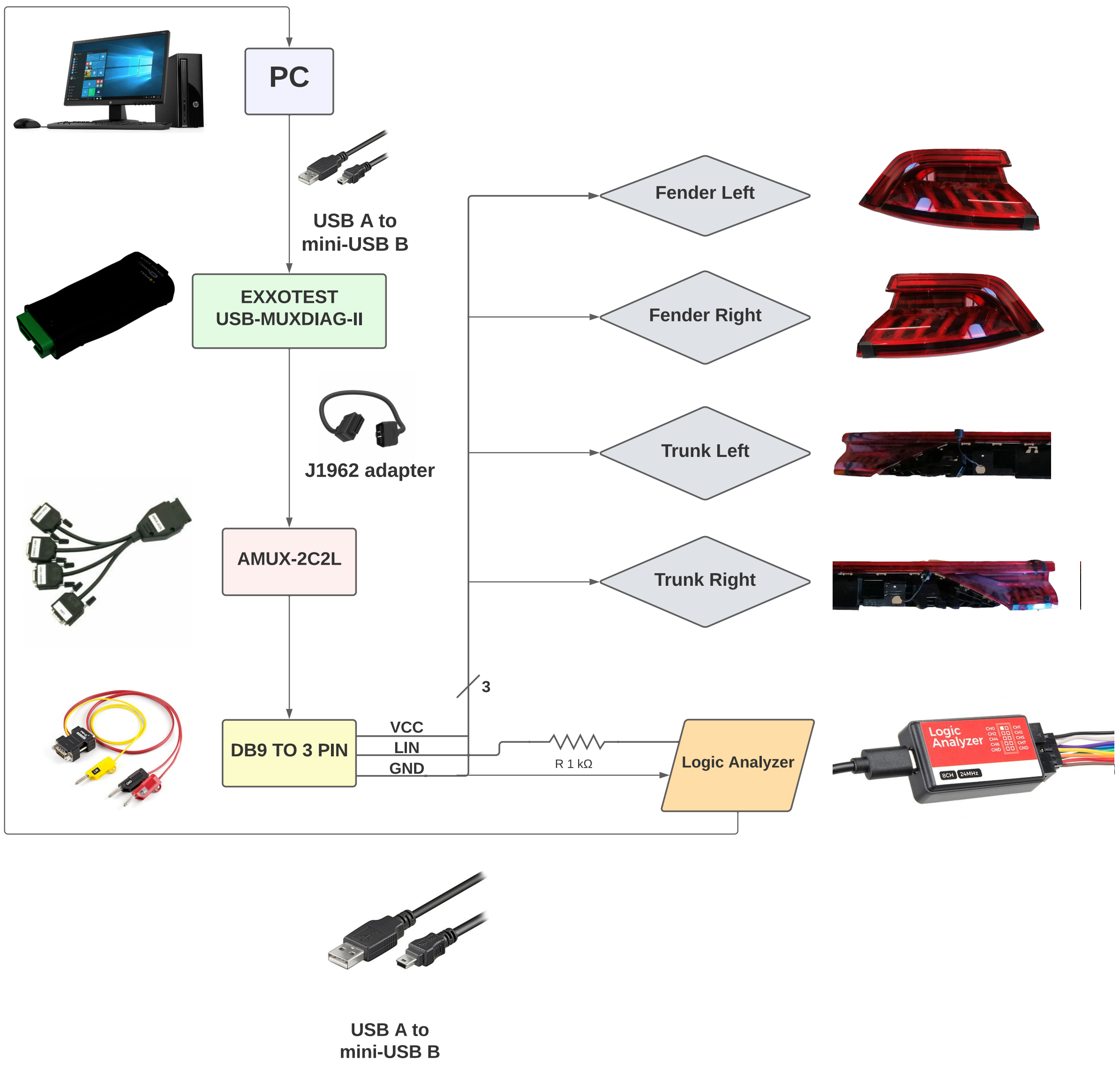
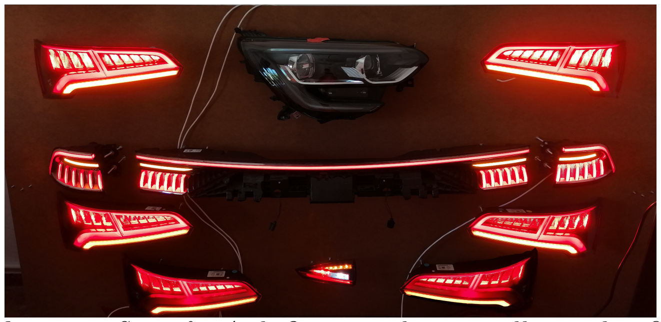


Figure 3.30 – *Connection diagram for LIN bus sniffing*



Video 3.1 – *Setup for Audi Q8 2018 pilots controlling and sniffing*

3.1 is a video with an example of using the "*Audi Rearlamp Management v1.1.13*" software in conjunction with the "*Exxotest USB-MUXDIAG-II*" and the different connectors used to perform the sniffing. 3.2 is also another one where the connection method illustrated in the previous diagram is shown.

Video 3.2 – *Controling Audi Q8 2018 pilots with Audi "Rearlamps Management v1.1.13"*

3.4.3 LIN bus sniffing

3.4.3.1 Introduction

The frames corresponding to LIN protocol that the *"Exxotest USB-MUXDIAG-II"* device in subsubsection 3.4.1.2 controlled by the software "AUDI Rearlamps Management v1.1.13" in subsubsection 3.4.1.1 sends through the LIN bus are transparent to the user, that is, the exact LIN frame that corresponds to the function of the selected pilot is not shown at any time in the Software.

That is why our goal in this section is to find out:

1. The **frame schedule** the software follows to keep the pilot running.

2. The **signals** corresponding to the activation of each function, that is, the values of the data field that the master must modify to activate the different functions.

For our work, we would only be interested in knowing the nature of the control frames, whose signals we will have to program to be written by the master, as they are the ones that will allow us to activate and/or deactivate the functions that we want from the LIN pilots. However, it will also be useful to analyze the defect diagnosis frames.

3.4.3.2 LIN Frames

As explained in subsection 3.4.4. When we look at the data table obtained by sniffing the LIN bus, we realize that not only one frame is sent to light a pilot function, but that the master node (the PC in our case) follows a frame routine, which we will call "LIN Schedule".

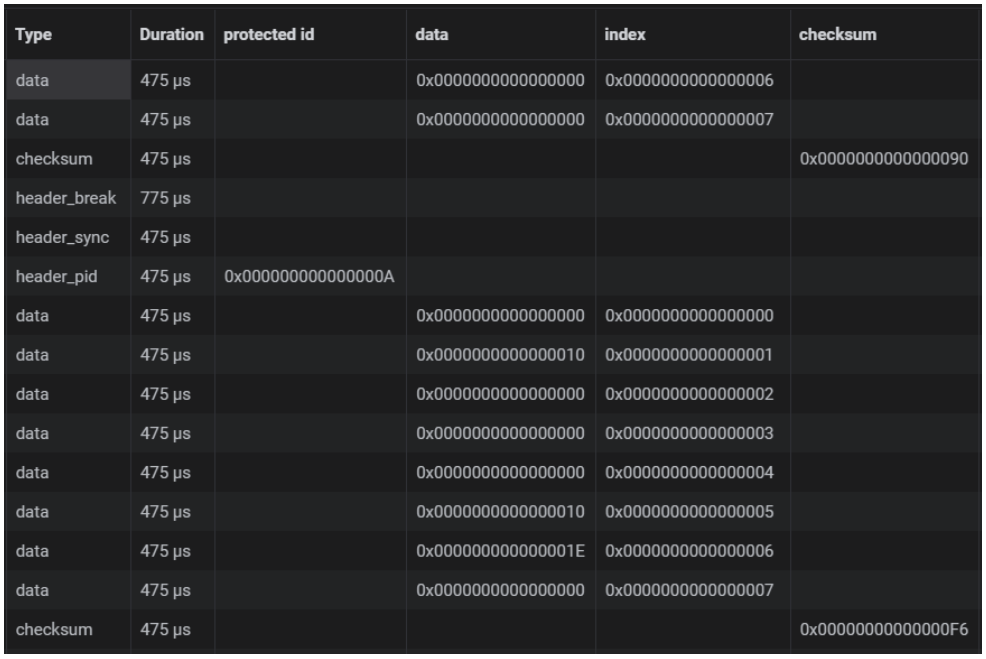
This section will explain the peculiarities of the schedule followed by the master node for the correct operation of the pilots, as well as the types of frames that exist in the communication.

Frames are sent and received every 10 ms and we have been able to identify up to 6 types of them. In the following sections we describe each one in detail.

3.4.3.2.1 Control frames

We call them ***"BCM2_SBBRe_01"***. They **have ID = 0x0A**. The most logical thing is that the signals that it carries in its data field (**8 bytes long**) are written by the master, since they represent the information necessary to "tell" each pilot (slaves) which functions to turn on or off.

Below is a real example of a frame of this type sniffed. As we can see, it follows the structure of the LIN frame seen in A.3.2



Type	Duration	protected id	data	index	checksum
data	475 µs		0x0000000000000000	0x0000000000000006	
data	475 µs		0x0000000000000000	0x0000000000000007	
checksum	475 µs				0x0000000000000090
header_break	775 µs				
header_sync	475 µs				
header_pid	475 µs	0x000000000000000A			
data	475 µs		0x0000000000000000	0x0000000000000000	
data	475 µs		0x0000000000000010	0x0000000000000001	
data	475 µs		0x0000000000000000	0x0000000000000002	
data	475 µs		0x0000000000000000	0x0000000000000003	
data	475 µs		0x0000000000000000	0x0000000000000004	
data	475 µs		0x0000000000000010	0x0000000000000005	
data	475 µs		0x000000000000001E	0x0000000000000006	
data	475 µs		0x0000000000000000	0x0000000000000007	
checksum	475 µs				0x00000000000000F6

Figure 3.31 – *Example of an actual control frame sniffed*

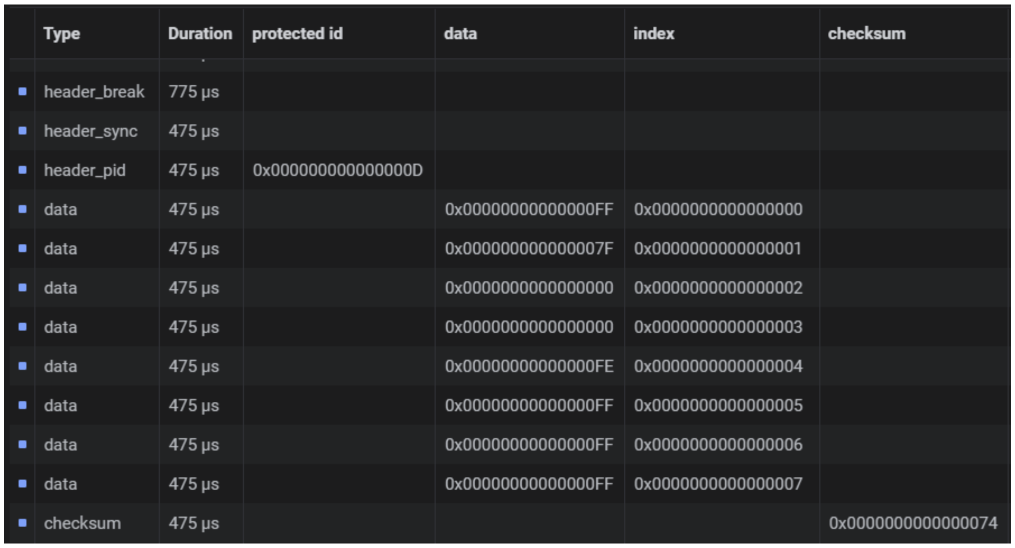
3.4.3.2.2 Defect diagnosis frames

4 different frames have been identified with IDs ranging from 0x0B to 0x0E, each of them corresponding to the diagnosis of errors/failures of one of the 4 LIN pilots (3.4.1.3) that we have:

- ***"SHL_li2_01 (ID:0x0B)"***: Trunk Left
- ***"SHL_re2_01 (ID:0x0C)"***: Trunk Right
- ***"SHL_li1_01 (ID:0x0D)"***: Fender Left
- ***"SHL_re1_01 (ID:0x0E)"***: Fender Right

In this case, it seems logical that the signals carried by these frames in their data field (**8 bytes long**) are **generated by the corresponding slave pilot** and destined for the master. In this way, the user with access to the master can **read the failure diagnosis**.

Below we can see an example of a fault diagnosis frame with signals written by the "Fender Left (Figure 3.23 (a))" pilot ("*SHL_li1_01 (ID:0x0D)*")



	Type	Duration	protected id	data	index	checksum
■	header_break	775 µs				
■	header_sync	475 µs				
■	header_pid	475 µs	0x000000000000000D			
■	data	475 µs		0x00000000000000FF	0x0000000000000000	
■	data	475 µs		0x000000000000007F	0x0000000000000001	
■	data	475 µs		0x0000000000000000	0x0000000000000002	
■	data	475 µs		0x0000000000000000	0x0000000000000003	
■	data	475 µs		0x00000000000000FE	0x0000000000000004	
■	data	475 µs		0x00000000000000FF	0x0000000000000005	
■	data	475 µs		0x00000000000000FF	0x0000000000000006	
■	data	475 µs		0x00000000000000FF	0x0000000000000007	
■	checksum	475 µs				0x0000000000000074

Figure 3.32 – *Example of an actual defect diagnosis frame sniffed*

We will not pay too much attention to the signals present in this type of frames, since, being fault diagnosis frames, changing the value of the signals is not as easy as with control frames 3.4.3.2.1, in fact, we do not see any way to do it

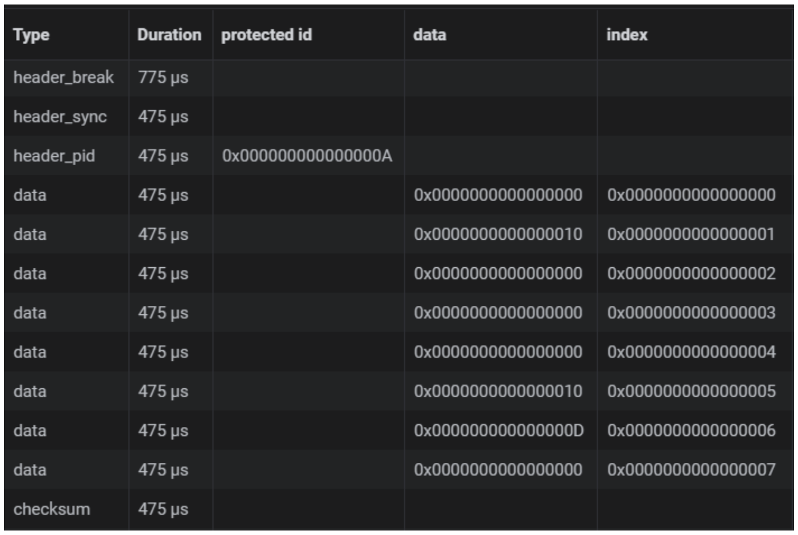
3.4.3.3 LIN Signals

In this section we will try, as far as possible, to identify the different signals present in the frames described in subsubsection 3.4.3.2.

To do this, we will activate/deactivate each function separately from the *"AUDI Rearlamps Management v1.1.13"* software which functioning is explained in subsubsection 3.4.1.1 and observe which bits of the data field react to the changes in each case.

Through this observation we will draw a conclusion about the position of each signal in the data field of the control frame explained in paragraph 3.4.3.2.1

3.4.3.3.1 Static tail



Type	Duration	protected id	data	index
header_break	775 µs			
header_sync	475 µs			
header_pid	475 µs	0x000000000000000A		
data	475 µs		0x0000000000000000	0x0000000000000000
data	475 µs		0x0000000000000010	0x0000000000000001
data	475 µs		0x0000000000000000	0x0000000000000002
data	475 µs		0x0000000000000000	0x0000000000000003
data	475 µs		0x0000000000000000	0x0000000000000004
data	475 µs		0x0000000000000010	0x0000000000000005
data	475 µs		0x000000000000000D	0x0000000000000006
data	475 µs		0x0000000000000000	0x0000000000000007
checksum	475 µs			

Figure 3.33 – *Control frame with static tail function ON*

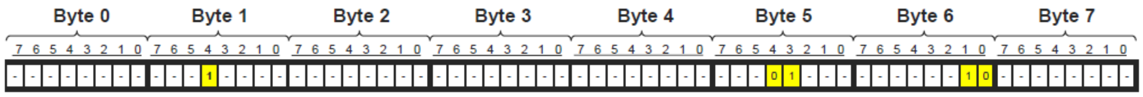
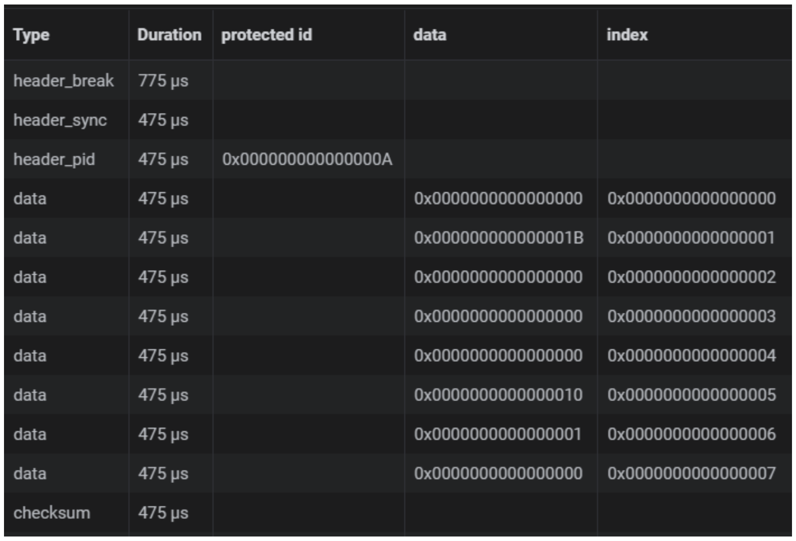


Figure 3.34 – *Data field with static tail function ON*



Type	Duration	protected id	data	index
header_break	775 µs			
header_sync	475 µs			
header_pid	475 µs	0x000000000000000A		
data	475 µs		0x0000000000000000	0x0000000000000000
data	475 µs		0x000000000000001B	0x0000000000000001
data	475 µs		0x0000000000000000	0x0000000000000002
data	475 µs		0x0000000000000000	0x0000000000000003
data	475 µs		0x0000000000000000	0x0000000000000004
data	475 µs		0x0000000000000010	0x0000000000000005
data	475 µs		0x0000000000000001	0x0000000000000006
data	475 µs		0x0000000000000000	0x0000000000000007
checksum	475 µs			

Figure 3.35 – *Control frame with static tail function OFF*

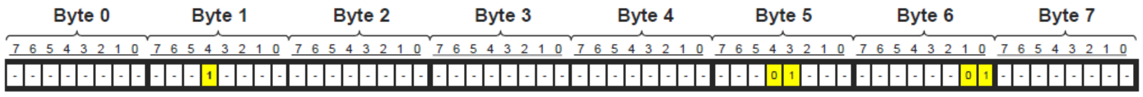
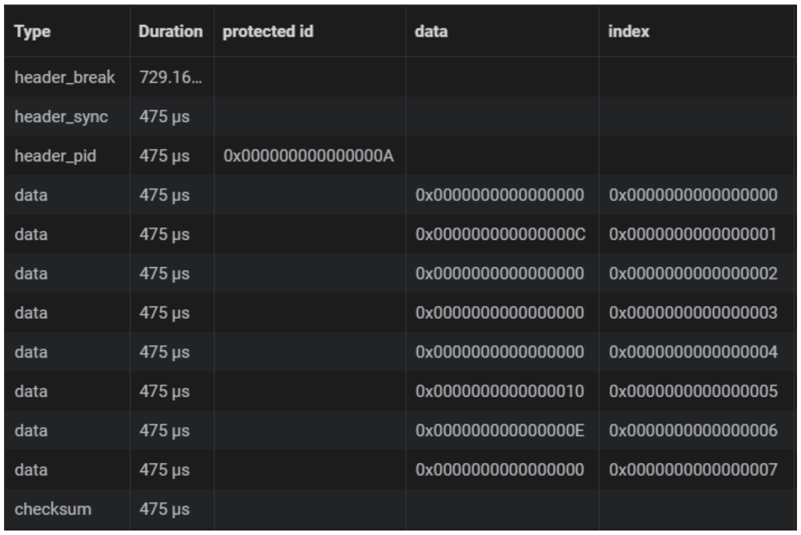


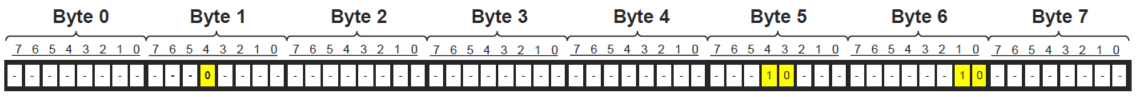
Figure 3.36 – *Data field with statil tail function OFF*

3.4.3.3.2 Animated tail



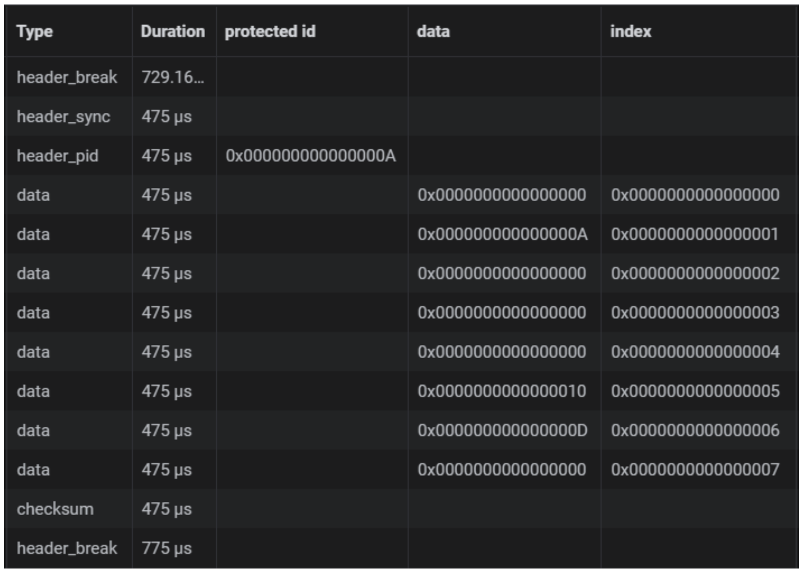
Type	Duration	protected id	data	index
header_break	729.16...			
header_sync	475 µs			
header_pid	475 µs	0x000000000000000A		
data	475 µs		0x0000000000000000	0x0000000000000000
data	475 µs		0x000000000000000C	0x0000000000000001
data	475 µs		0x0000000000000000	0x0000000000000002
data	475 µs		0x0000000000000000	0x0000000000000003
data	475 µs		0x0000000000000000	0x0000000000000004
data	475 µs		0x0000000000000010	0x0000000000000005
data	475 µs		0x000000000000000E	0x0000000000000006
data	475 µs		0x0000000000000000	0x0000000000000007
checksum	475 µs			

Figure 3.37 – *Control frame with animated tail function ON*



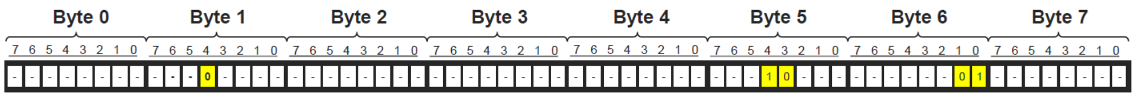
	7	6	5	4	3	2	1	0
Byte 0	-	-	-	-	-	-	-	-
Byte 1	-	-	-	0	-	-	-	-
Byte 2	-	-	-	-	-	-	-	-
Byte 3	-	-	-	-	-	-	-	-
Byte 4	-	-	-	-	-	-	-	-
Byte 5	-	-	-	1	0	-	-	-
Byte 6	-	-	-	-	-	-	1	0
Byte 7	-	-	-	-	-	-	-	-

Figure 3.38 – *Data field with animated tail function ON*



Type	Duration	protected id	data	index
header_break	729.16...			
header_sync	475 µs			
header_pid	475 µs	0x000000000000000A		
data	475 µs		0x0000000000000000	0x0000000000000000
data	475 µs		0x000000000000000A	0x0000000000000001
data	475 µs		0x0000000000000000	0x0000000000000002
data	475 µs		0x0000000000000000	0x0000000000000003
data	475 µs		0x0000000000000000	0x0000000000000004
data	475 µs		0x0000000000000010	0x0000000000000005
data	475 µs		0x000000000000000D	0x0000000000000006
data	475 µs		0x0000000000000000	0x0000000000000007
checksum	475 µs			
header_break	775 µs			

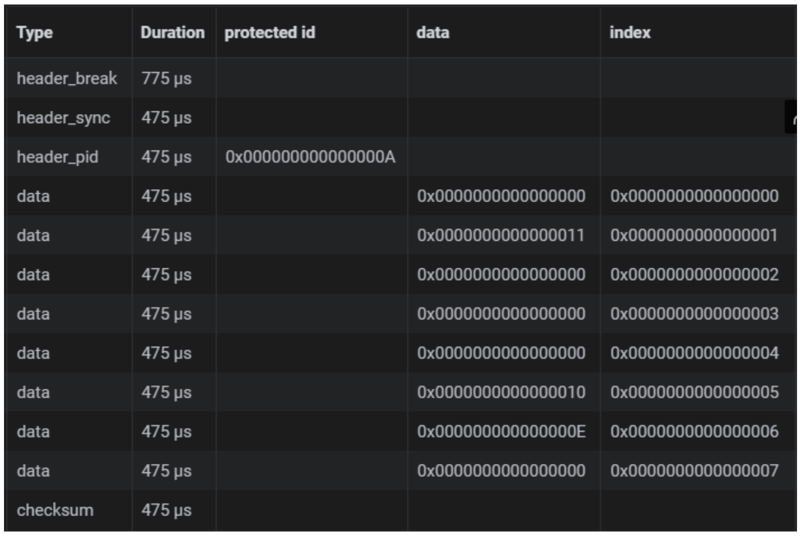
Figure 3.39 – *Control frame with animated tail function OFF*



	7	6	5	4	3	2	1	0
Byte 0	-	-	-	-	-	-	-	-
Byte 1	-	-	-	0	-	-	-	-
Byte 2	-	-	-	-	-	-	-	-
Byte 3	-	-	-	-	-	-	-	-
Byte 4	-	-	-	-	-	-	-	-
Byte 5	-	-	-	1	0	-	-	-
Byte 6	-	-	-	-	-	-	0	1
Byte 7	-	-	-	-	-	-	-	-

Figure 3.40 – *Data field with animated tail function OFF*

3.4.3.3.3 Dynamic tail



Type	Duration	protected id	data	index
header_break	775 µs			
header_sync	475 µs			
header_pid	475 µs	0x000000000000000A		
data	475 µs		0x0000000000000000	0x0000000000000000
data	475 µs		0x0000000000000011	0x0000000000000001
data	475 µs		0x0000000000000000	0x0000000000000002
data	475 µs		0x0000000000000000	0x0000000000000003
data	475 µs		0x0000000000000000	0x0000000000000004
data	475 µs		0x0000000000000010	0x0000000000000005
data	475 µs		0x000000000000000E	0x0000000000000006
data	475 µs		0x0000000000000000	0x0000000000000007
checksum	475 µs			

Figure 3.41 – *Control frame with dynamic tail function ON*

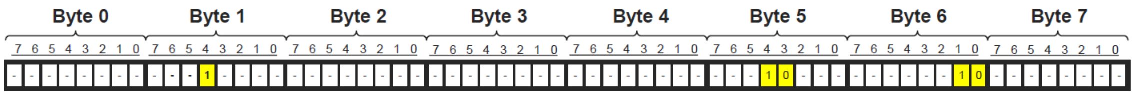
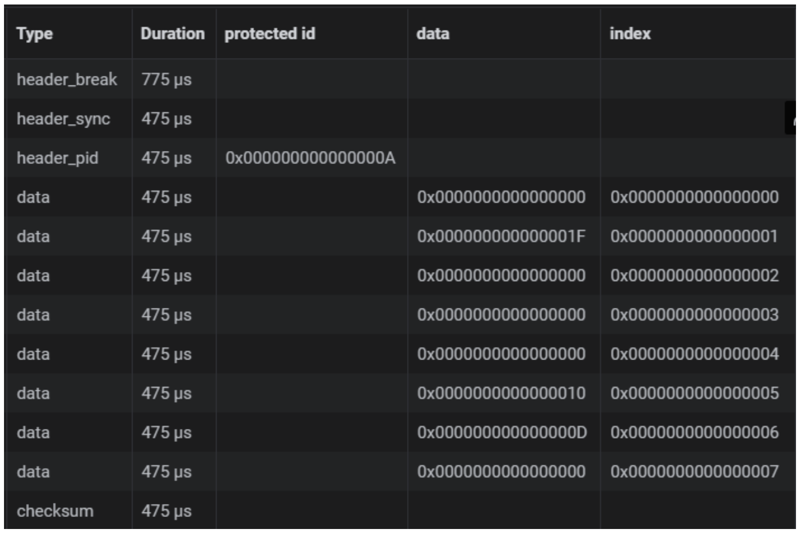


Figure 3.42 – *Data field with dynamic tail function ON*



Type	Duration	protected id	data	index
header_break	775 µs			
header_sync	475 µs			
header_pid	475 µs	0x000000000000000A		
data	475 µs		0x0000000000000000	0x0000000000000000
data	475 µs		0x000000000000001F	0x0000000000000001
data	475 µs		0x0000000000000000	0x0000000000000002
data	475 µs		0x0000000000000000	0x0000000000000003
data	475 µs		0x0000000000000000	0x0000000000000004
data	475 µs		0x0000000000000010	0x0000000000000005
data	475 µs		0x000000000000000D	0x0000000000000006
data	475 µs		0x0000000000000000	0x0000000000000007
checksum	475 µs			

Figure 3.43 – *Control frame with dynamic tail function ON*

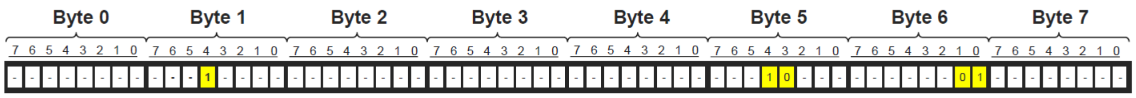
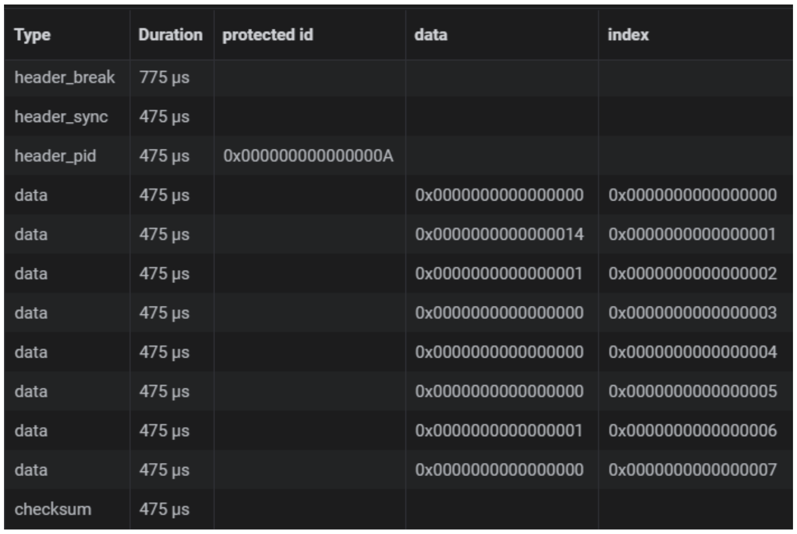


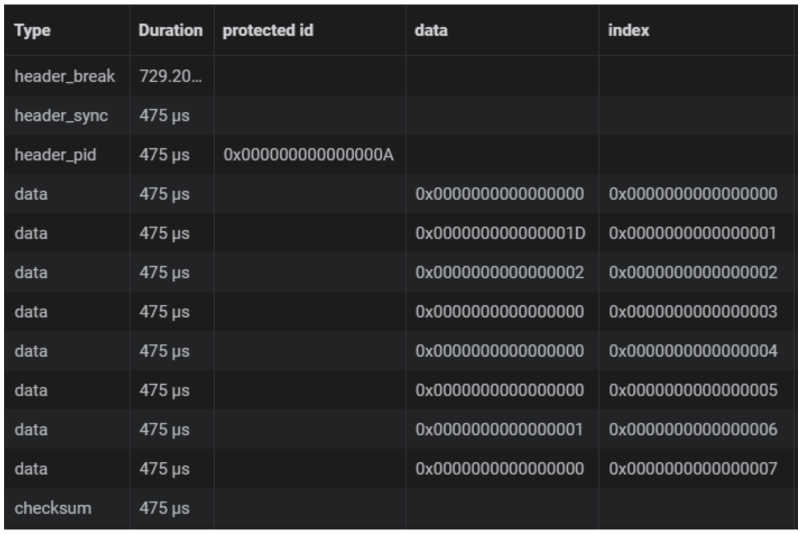
Figure 3.44 – *Data field with animated tail function OFF*

3.4.3.3.4 Turn indicator



Type	Duration	protected id	data	index
header_break	775 µs			
header_sync	475 µs			
header_pid	475 µs	0x000000000000000A		
data	475 µs		0x0000000000000000	0x0000000000000000
data	475 µs		0x0000000000000014	0x0000000000000001
data	475 µs		0x0000000000000001	0x0000000000000002
data	475 µs		0x0000000000000000	0x0000000000000003
data	475 µs		0x0000000000000000	0x0000000000000004
data	475 µs		0x0000000000000000	0x0000000000000005
data	475 µs		0x0000000000000001	0x0000000000000006
data	475 µs		0x0000000000000000	0x0000000000000007
checksum	475 µs			

Figure 3.45 – *Control frame with left TI running function OFF*



Type	Duration	protected id	data	index
header_break	729.20...			
header_sync	475 µs			
header_pid	475 µs	0x000000000000000A		
data	475 µs		0x0000000000000000	0x0000000000000000
data	475 µs		0x000000000000001D	0x0000000000000001
data	475 µs		0x0000000000000002	0x0000000000000002
data	475 µs		0x0000000000000000	0x0000000000000003
data	475 µs		0x0000000000000000	0x0000000000000004
data	475 µs		0x0000000000000000	0x0000000000000005
data	475 µs		0x0000000000000001	0x0000000000000006
data	475 µs		0x0000000000000000	0x0000000000000007
checksum	475 µs			

Figure 3.46 – *Control frame with right TI running function OFF*

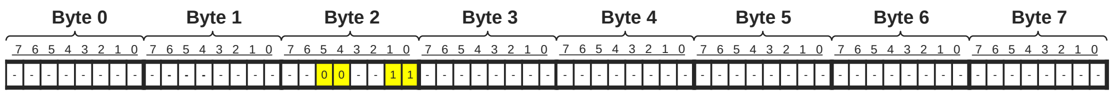
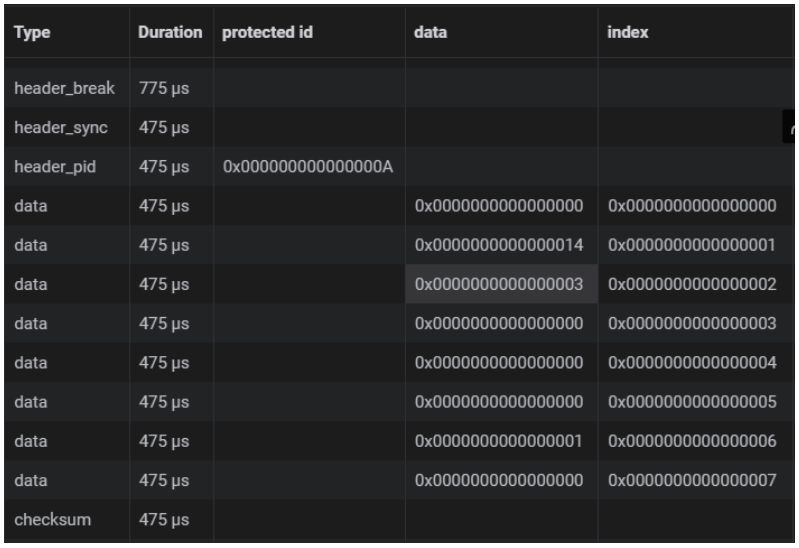


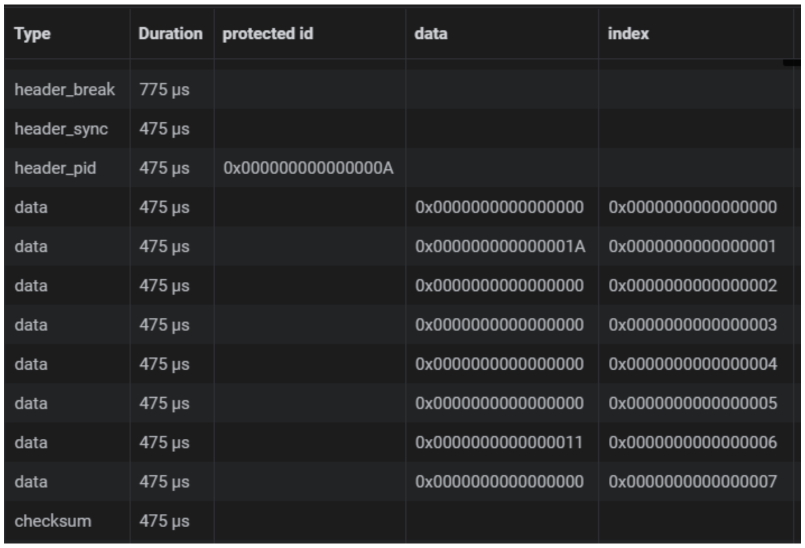
Figure 3.47 – *Data field with both running TI function OFF*



Type	Duration	protected id	data	index
header_break	775 µs			
header_sync	475 µs			
header_pid	475 µs	0x000000000000000A		
data	475 µs		0x0000000000000000	0x0000000000000000
data	475 µs		0x0000000000000014	0x0000000000000001
data	475 µs		0x0000000000000003	0x0000000000000002
data	475 µs		0x0000000000000000	0x0000000000000003
data	475 µs		0x0000000000000000	0x0000000000000004
data	475 µs		0x0000000000000000	0x0000000000000005
data	475 µs		0x0000000000000001	0x0000000000000006
data	475 µs		0x0000000000000000	0x0000000000000007
checksum	475 µs			

Figure 3.48 – *Control frame with both TI running function OFF*

3.4.3.3.5 Reverse



Type	Duration	protected id	data	index
header_break	775 µs			
header_sync	475 µs			
header_pid	475 µs	0x000000000000000A		
data	475 µs		0x0000000000000000	0x0000000000000000
data	475 µs		0x000000000000001A	0x0000000000000001
data	475 µs		0x0000000000000000	0x0000000000000002
data	475 µs		0x0000000000000000	0x0000000000000003
data	475 µs		0x0000000000000000	0x0000000000000004
data	475 µs		0x0000000000000000	0x0000000000000005
data	475 µs		0x0000000000000011	0x0000000000000006
data	475 µs		0x0000000000000000	0x0000000000000007
checksum	475 µs			

Figure 3.49 – *Control frame with reverse function ON*

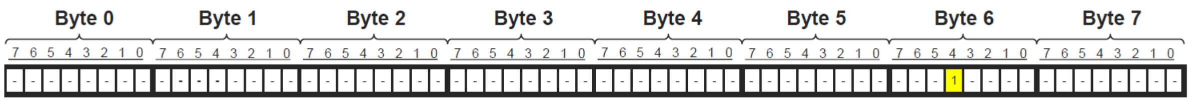
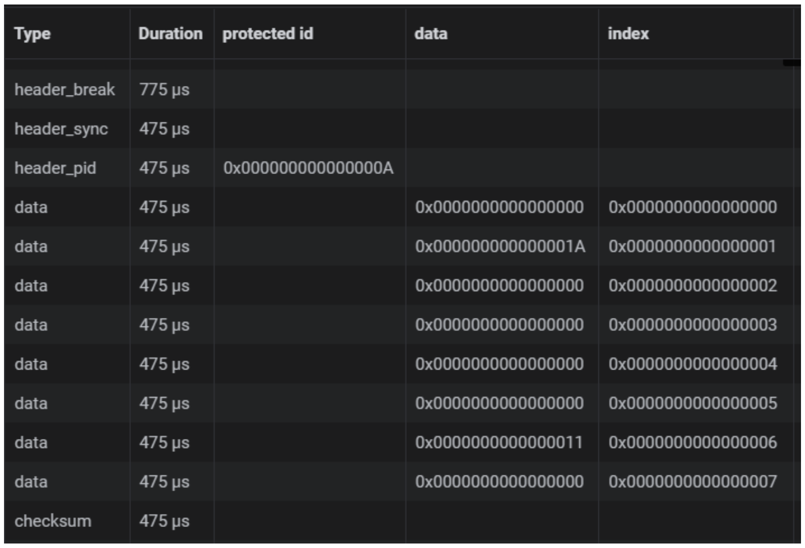


Figure 3.50 – *Data field with reverse function ON*



Type	Duration	protected id	data	index
header_break	775 µs			
header_sync	475 µs			
header_pid	475 µs	0x000000000000000A		
data	475 µs		0x0000000000000000	0x0000000000000000
data	475 µs		0x000000000000001A	0x0000000000000001
data	475 µs		0x0000000000000000	0x0000000000000002
data	475 µs		0x0000000000000000	0x0000000000000003
data	475 µs		0x0000000000000000	0x0000000000000004
data	475 µs		0x0000000000000000	0x0000000000000005
data	475 µs		0x0000000000000011	0x0000000000000006
data	475 µs		0x0000000000000000	0x0000000000000007
checksum	475 µs			

Figure 3.51 – *Control frame with reverse function OFF*

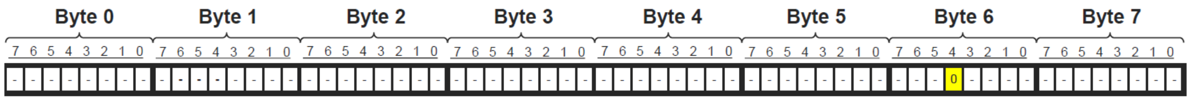
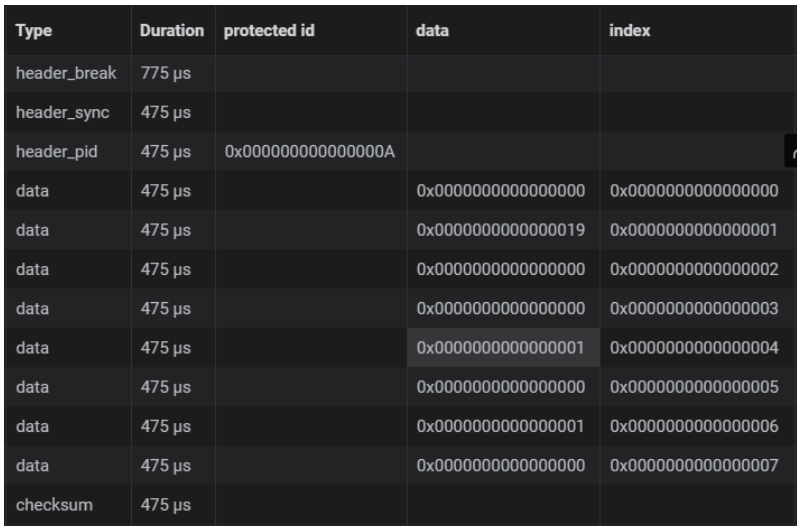


Figure 3.52 – *Data field with reverse function OFF*

3.4.3.3.6 Stop



Type	Duration	protected id	data	index
header_break	775 µs			
header_sync	475 µs			
header_pid	475 µs	0x000000000000000A		
data	475 µs		0x0000000000000000	0x0000000000000000
data	475 µs		0x0000000000000019	0x0000000000000001
data	475 µs		0x0000000000000000	0x0000000000000002
data	475 µs		0x0000000000000000	0x0000000000000003
data	475 µs		0x0000000000000001	0x0000000000000004
data	475 µs		0x0000000000000000	0x0000000000000005
data	475 µs		0x0000000000000001	0x0000000000000006
data	475 µs		0x0000000000000000	0x0000000000000007
checksum	475 µs			

Figure 3.53 – *Control frame with STOP function ON*

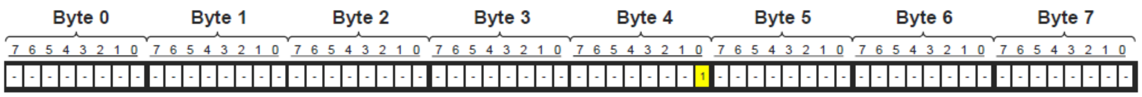
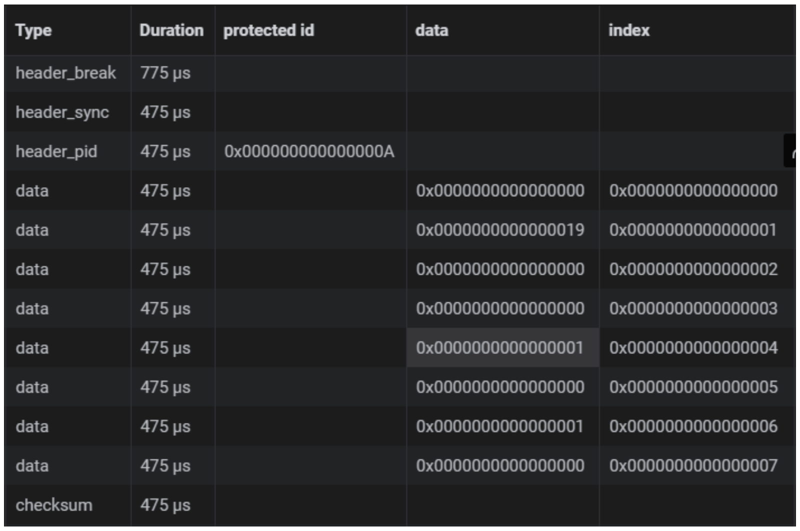


Figure 3.54 – *Data field with STOP function ON*



Type	Duration	protected id	data	index
header_break	775 µs			
header_sync	475 µs			
header_pid	475 µs	0x000000000000000A		
data	475 µs		0x0000000000000000	0x0000000000000000
data	475 µs		0x0000000000000019	0x0000000000000001
data	475 µs		0x0000000000000000	0x0000000000000002
data	475 µs		0x0000000000000000	0x0000000000000003
data	475 µs		0x0000000000000001	0x0000000000000004
data	475 µs		0x0000000000000000	0x0000000000000005
data	475 µs		0x0000000000000001	0x0000000000000006
data	475 µs		0x0000000000000000	0x0000000000000007
checksum	475 µs			

Figure 3.55 – *Control frame with STOP function OFF*

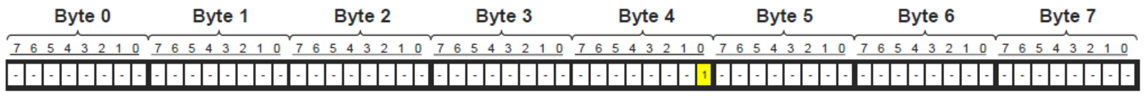


Figure 3.56 – *Data field with STOP function OFF*

3.4.4 LIN Schedule

Once we have identified all the frames in subsubsection 3.4.3.2 present in the LIN communication and the corresponding signals in subsubsection 3.4.3.3 that they carry in their data field, we will proceed to obtain the LIN schedule, which determines the order followed cyclically in which the master sends the frames through the cluster.

The frames follow the schedule in Figure 3.57 (a). As we can see, the value of the ID to which the fault diagnosis frame explained in paragraph 3.4.3.2.2 is addressed is increased by 1 every time this frame is sent from 0x0B until it reaches the value of 0x0E. In this way, the master sweeps the IDs of the different slaves (each of the 4 LIN pilots showed in subsubsection 3.4.1.3) which will respond when their turn comes with the signals corresponding to the errors / failures they have.

In addition, every 20 ms the master sends a control frame to which **it responds itself**, since it headlight it wants to turn ON or OFF. Something that we have also realized is that every time a control frame (paragraph 3.4.3.2.1) is sent, the value of a signal that we will call ***"BZ"*** is updated. This signal's location in the data field of the control frame is shown in Figure 3.57 (a). This signal cyclically follows the pattern shown in Figure 3.58 (b).

Following this schedule is mandatory for the successful operation of the pilot.

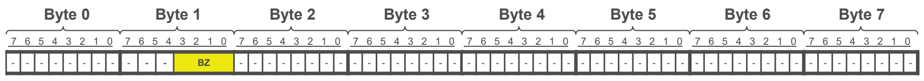
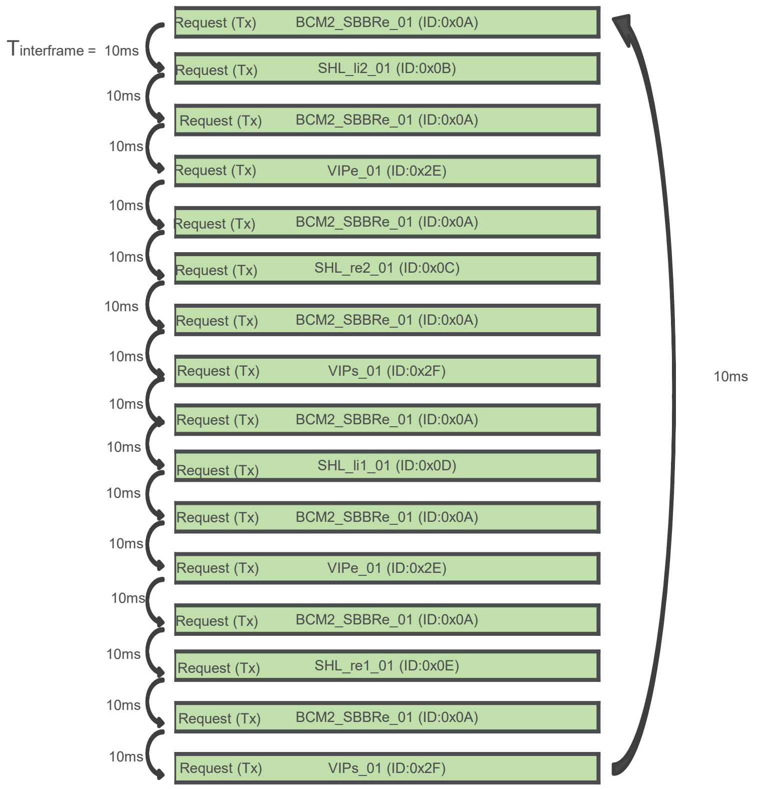
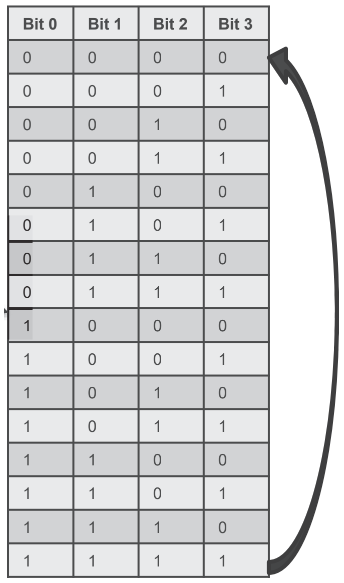


Figure 3.57 – *Location of the BZ signal in the LIN frame data field*


Tinterframe = 10ms
Request (Tx) BCM2_SBBRe_01 (ID:0x0A)
Request (Tx) SHL_li2_01 (ID:0x0B)
Request (Tx) BCM2_SBBRe_01 (ID:0x0A)
Request (Tx) VIPe_01 (ID:0x2E)
Request (Tx) BCM2_SBBRe_01 (ID:0x0A)
Request (Tx) SHL_re2_01 (ID:0x0C)
Request (Tx) BCM2_SBBRe_01 (ID:0x0A)
Request (Tx) VIPs_01 (ID:0x2F)
Request (Tx) BCM2_SBBRe_01 (ID:0x0A)
Request (Tx) SHL_li1_01 (ID:0x0D)
Request (Tx) BCM2_SBBRe_01 (ID:0x0A)
Request (Tx) VIPe_01 (ID:0x2E)
Request (Tx) BCM2_SBBRe_01 (ID:0x0A)
Request (Tx) SHL_re1_01 (ID:0x0E)
Request (Tx) BCM2_SBBRe_01 (ID:0x0A)
Request (Tx) VIPs_01 (ID:0x2F)
10ms
10ms
10ms
10ms
10ms
10ms
10ms
10ms
10ms
10ms
10ms
10ms
10ms
10ms
10ms
10ms

(a) *LIN Schedule table*



Bit 0	Bit 1	Bit 2	Bit 3
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

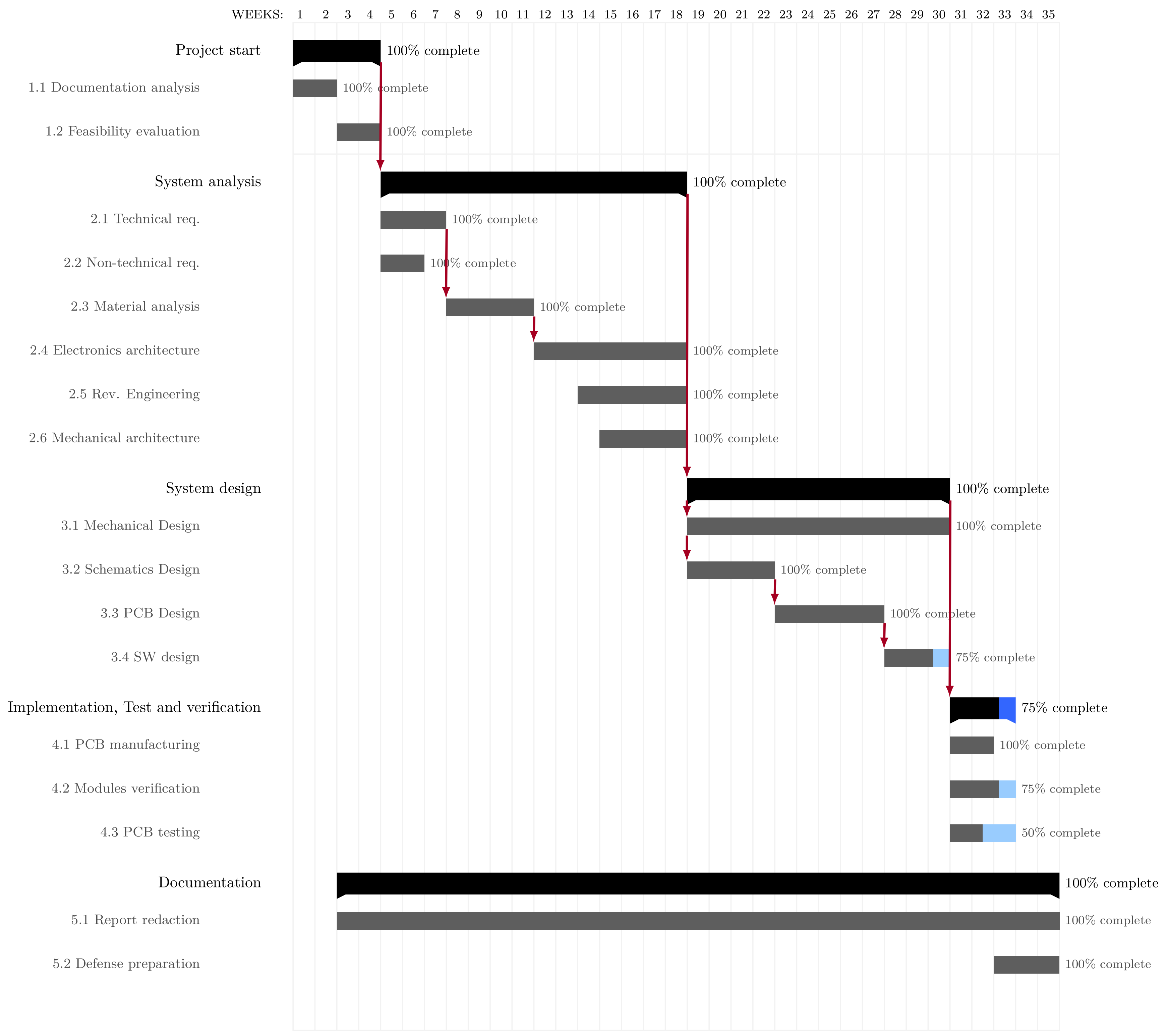
(b) *BZ signal sequence*

Figure 3.58 – *Final LIN schedule table followed by master*

3.5 Gantt diagram

In this section we are going to be showing a **Gantt diagram**. This is one of the most popular and useful ways of showing activities (tasks or events) displayed against time. On the left of the chart there is a list of the activities and along the top is a suitable time scale. Each activity is represented by a bar; the position and length of the bar reflects the start date, duration and end date of the activity.

For the case of our project, we will show the distribution of time that has been made to carry out the different tasks required during the development of the project.


WEEKS: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
Project start
100% complete
1.1 Documentation analysis
100% complete
1.2 Feasibility evaluation
100% complete
System analysis
100% complete
2.1 Technical req.
100% complete
2.2 Non-technical req.
100% complete
2.3 Material analysis
100% complete
2.4 Electronics architecture
100% complete
2.5 Rev. Engineering
100% complete
2.6 Mechanical architecture
100% complete
System design
100% complete
3.1 Mechanical Design
100% complete
3.2 Schematics Design
100% complete
3.3 PCB Design
100% complete
3.4 SW design
75% complete
Implementation, Test and verification
75% complete
4.1 PCB manufacturing
100% complete
4.2 Modules verification
75% complete
4.3 PCB testing
50% complete
Documentation
100% complete
5.1 Report redaction
100% complete
5.2 Defense preparation
100% complete

Chapter 4

System Design

After the different stages of analysis and requirements definition performed in former chapters, this chapter will address the **system design**.

We will see more in detail the **specifications of each module** chosen in section 3.2 and its nature to know how to connect it correctly to the rest of modules. We will also study **the way in which we must connect all the modules with each other** so that the PCB performs its task correctly at the schematic level, justifying each design decision.

Finally we will also emphasize the **step-by-step design of the PCB**, including the **mechanical design** part, the placement of the different modules and the routing of the same.

For all this we will use Altium Designer® 21, a software created for PCB design in an intuitive way.

4.1 Electronic Design and component descriptions

Altium Designer® 21 makes possible the direct and instant relationship between a schematic that we have designed and the corresponding PCB, relating the *.SCH* and *.PCB* files. So in this section we will take care of the design of both.

4.1.1 Schematics Design

In this section we will focus on the **design of the schematic file (*.SCH*)**, showing the final circuits that each module selected in chapter 4 requires for its correct operation.

To do this, we will first need to know more about the specifications of the components. So this section will follow a structure similar to section 3.2 but keeping in mind that we have already **made the choice** about which component will be in charge of performing **the main function of each module**.

4.1.1.1 Microcontroller module

As we conclude in subsubsection 3.2.1.2, the microcontroller that we will be using is the Arduino UNO. In Figure 4.1, Figure 4.2 and Figure 4.3 we can see the complete pinout of Arduino UNO.

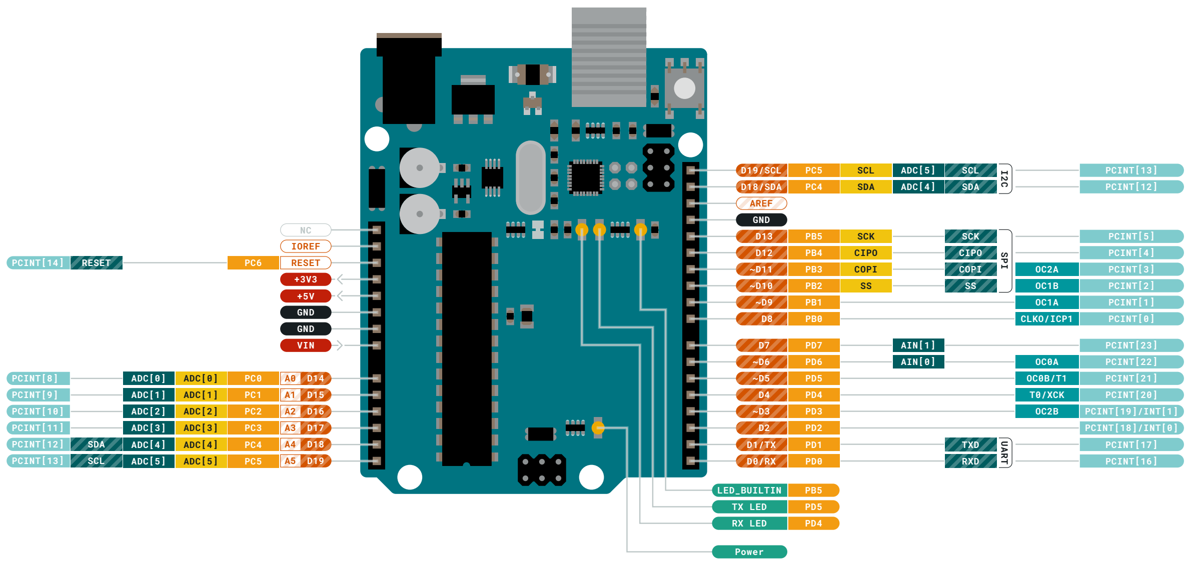

D19/SCL PC5 SCL ADC[5] SCL PCINT[13]
D18/SDA PC4 SDA ADC[4] SDA PCINT[12]
I2C
AREF
GND
D13 PB5 SCK SCK PCINT[5]
D12 PB4 CIPO CIPO PCINT[4]
~D11 PB3 COPI COPI OC2A PCINT[3]
~D10 PB2 SS SS OC1B PCINT[2]
SPI
~D9 PB1 OC1A PCINT[1]
D8 PB0 CLKO/ICP1 PCINT[0]
D7 PD7 AIN[1] PCINT[23]
~D6 PD6 AIN[0] OC0A PCINT[22]
~D5 PD5 OC0B/T1 PCINT[21]
D4 PD4 T0/XCK PCINT[20]
~D3 PD3 OC2B PCINT[19]/INT[1]
D2 PD2 PCINT[18]/INT[0]
D1/TX PD1 TXD PCINT[17]
D0/RX PD0 RXD PCINT[16]
UART
LED_BUILTIN PB5
TX LED PD5
RX LED PD4
Power
NC
IOREF
PCINT[14] RESET PC6 RESET
+3V3
+5V
GND
GND
VIN
PCINT[8] ADC[0] ADC[0] PC0 A0 D14
PCINT[9] ADC[1] ADC[1] PC1 A1 D15
PCINT[10] ADC[2] ADC[2] PC2 A2 D16
PCINT[11] ADC[3] ADC[3] PC3 A3 D17
PCINT[12] SDA ADC[4] ADC[4] PC4 A4 D18
PCINT[13] SCL ADC[5] ADC[5] PC5 A5 D19

Figure 4.1 – *Arduino UNO pinout (1)*

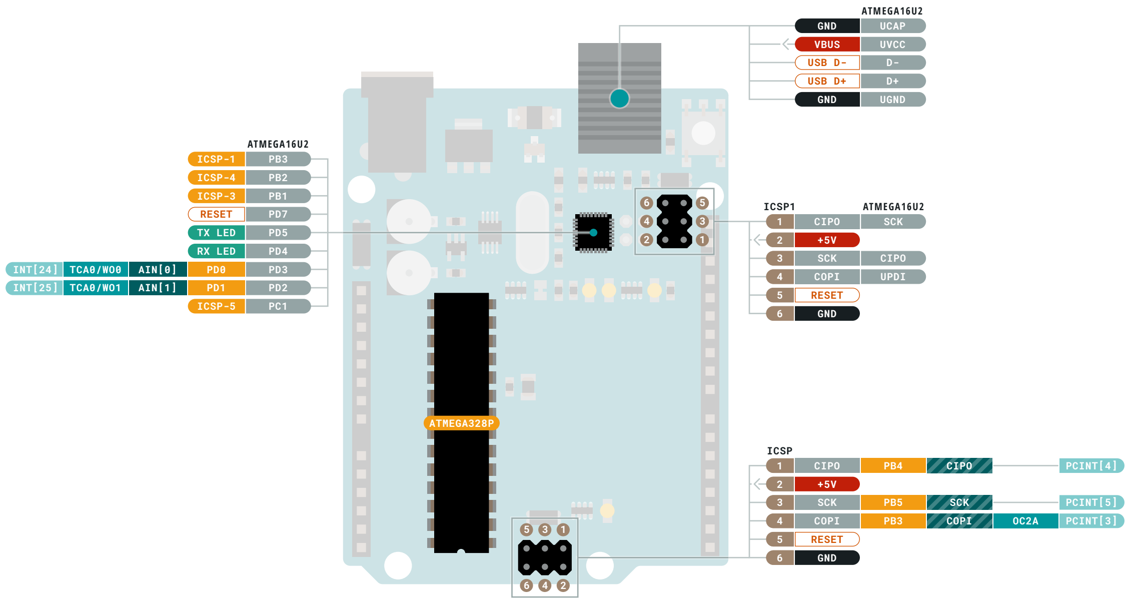


Figure 4.2 – *Arduino UNO pinout (2)*

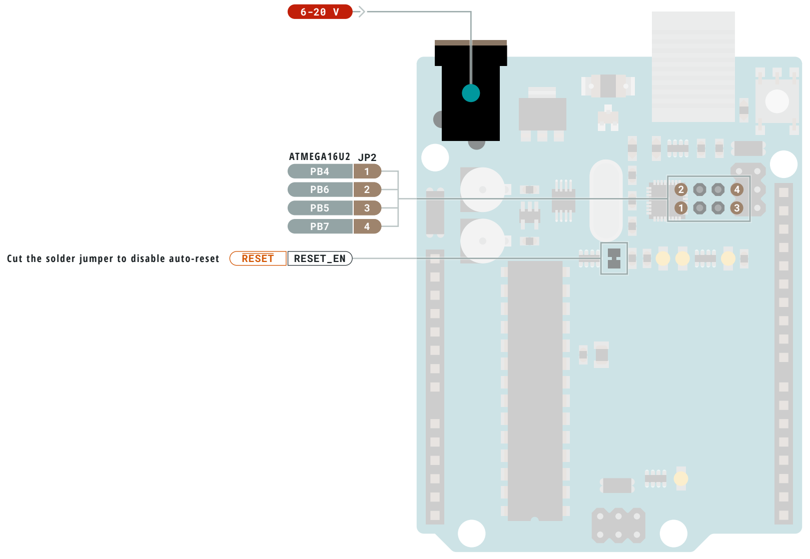


Figure 4.3 – *Arduino UNO pinout (3)*

Figure 4.4 shows the schematic symbol corresponding to the Arduino UNO created in Altium Designer® 21 and manually added to our custom library for later inclusion in the final schematic. For the case of the microcontroller, as we see, it is very important to design an Altium Designer® 21 schematic symbol that is very precise. This is because each Arduino UNO PIN has a multitude of different functions that must be made clear.

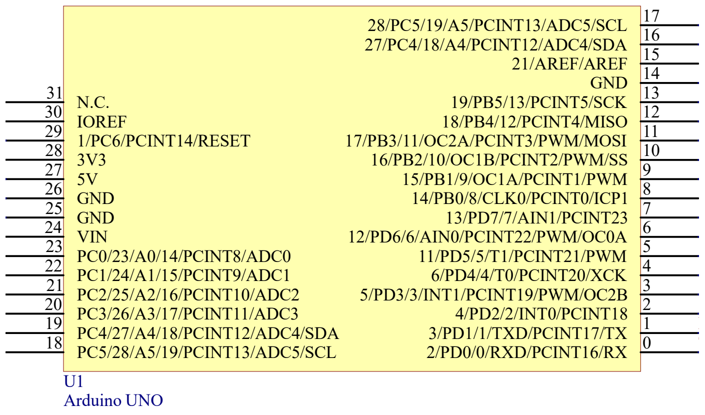

31 N.C.
30 IOREF
29 1/PC6/PCINT14/RESET
28 3V3
27 5V
26 GND
25 GND
24 VIN
23 PC0/23/A0/14/PCINT8/ADC0
22 PC1/24/A1/15/PCINT9/ADC1
21 PC2/25/A2/16/PCINT10/ADC2
20 PC3/26/A3/17/PCINT11/ADC3
19 PC4/27/A4/18/PCINT12/ADC4/SDA
18 PC5/28/A5/19/PCINT13/ADC5/SCL
28/PC5/19/A5/PCINT13/ADC5/SCL 17
27/PC4/18/A4/PCINT12/ADC4/SDA 16
21/AREF/AREF 15
GND 14
19/PB5/13/PCINT5/SCK 13
18/PB4/12/PCINT4/MISO 12
17/PB3/11/OC2A/PCINT3/PWM/MOSI 11
16/PB2/10/OC1B/PCINT2/PWM/SS 10
15/PB1/9/OC1A/PCINT1/PWM 9
14/PB0/8/CLK0/PCINT0/ICP1 8
13/PD7/7/AIN1/PCINT23 7
12/PD6/6/AIN0/PCINT22/PWM/OC0A 6
11/PD5/5/T1/PCINT21/PWM 5
6/PD4/4/T0/PCINT20/XCK 4
5/PD3/3/INT1/PCINT19/PWM/OC2B 3
4/PD2/2/INT0/PCINT18 2
3/PD1/1/TXD/PCINT17/TX 1
2/PD0/0/RXD/PCINT16/RX 0
U1
Arduino UNO

Figure 4.4 – *Arduino UNO Altium Designer® 21 schematic symbol*

Figure 4.5 shows the components and modules to which the Arduino UNO microcontroller has been connected for the applications in which we are interested. The following sections will go into detail about the modules mentioned.

- **Display module** must be connected to I2C port, which is available at PINs 16 and 17 of the Arduino UNO
- **Shift register module** must be connected to 3 digital PINs. We chose them to be PINs 13, 12, 11 and 9 of Arduino UNO.
- **Music player module** must be connected to a serial port. Since we have already used the default

serial port of the Arduino UNO to connect the USB to serial and LIN transceiver modules, we need to implement by software a VSP to connect this module. For this we have decided to use the PINs 7 and 6 of Arduino UNO.

- **USB to serial module** must be connected to the default serial port of the Arduino UNO, which is available at PINs 0 and 1. However, these pins are the only ones through which the microcontroller can be programmed externally without connecting the USB cable, so we might want to leave them free for that purpose. That is why we will use another VSP for this purpose at PINs 3 and 4. In addition, we have used the Arduino UNO's RESET PIN (available at PIN 29) to implement the reset of the USB to serial module.

- **LIN** transceiver module is connected to the same VSP as USB to serial module. This is because we want to implement the possibility of establishing LIN communication in two different ways: generating it in the Arduino UNO code or directly from a PC via USB. The problem that this entails is that, when one of the two communications is taking place, the other can interfere causing a collision between them. That is why we have decided to include a **jumper** in the thread corresponding to the TX. We will see its operation in subsubsection 4.1.1.5

- The IR receiver must be connected to a digital PIN. We decided to connect it to PIN 2.

- We also included a **general purpose push button** at PIN 5 with a 10 KΩ pull-down resistor. This value guarantees the low-level voltage at the digital PIN.

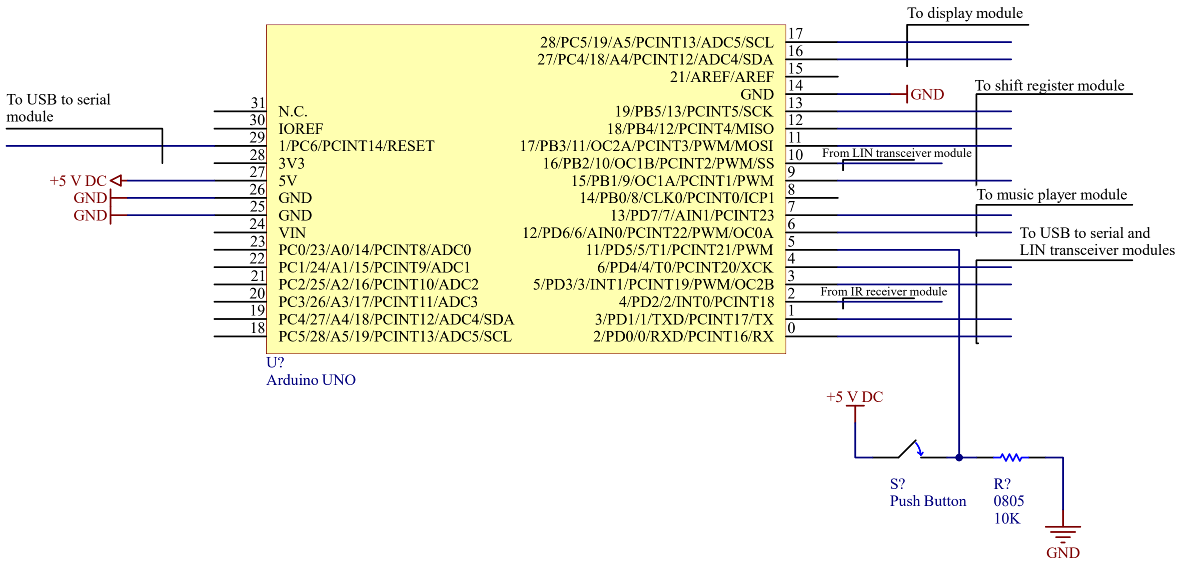


Figure 4.5 – *Arduino UNO Altium Designer® 21 application circuit*

It is important to note that the reason why each module must be connected to the pins that have been specified in this section is developed specifically for each module in the following sections.

4.1.1.2 Power Supply module

In subsection 3.2.2 we decided to use an ATX switched power supply to power our board. Figure 4.6 shows the pinout of the **24 pin Molex 39-01-2240 connector.**

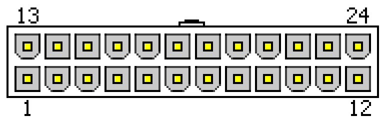


Figure 4.6 – *ATX switched power supply connector pinout*

In Table 4.1 the function and cable color of each of 14 PIN is described. As we can see, this power supply offers outputs at different voltage levels. In addition, these levels are just the ones that interest us for our design, Especially the 5 V and 12 V.

In Figure 4.8 we can see the ATX symbol created in Altium Designer® 21 and manually added to our custom library to be used in the product schematic. As we can see, this time the power supply symbol corresponds to **two terminal blocks** that we will use as **input**. One of them has a 5 and 12 V input, the other only 12 V. It has been necessary to do it this way because, as we said in subsection 3.2.11, the maximum current that a terminal block supports is **13.5 A**. For our application, this limitation would not cause any problem for the +5 V source, but it would for the +12 V source, which has to be prepared to be able to power **all the lights on at the same time** which implies a current of about **20 A** as we can see in section E.2. By dividing the +12 V input into two terminal blocks, we achieve that the current is divided in two, thus allowing operation within the established ranges.

In Figure 4.7 we can see the circuit that has been implemented to power the board. This is a very simple implementation. It has been considered opportune to add LEDs to offer a visual indication that each voltage level is correctly reaching the board. A red LED has been used to indicate the presence of the +12 V voltage and a blue one to indicate the +5 V.

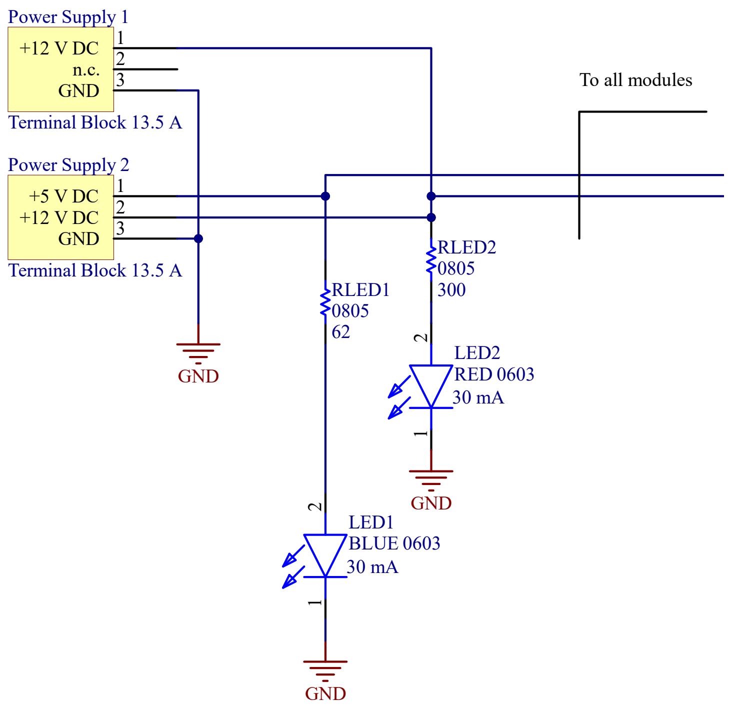


Figure 4.7 – *ATX switched power supply application circuit*

PIN	SYMBOL	COLOR	FUNCTION
1	3.3 V	Orange	3.3 V DC
2	3.3 V	Orange	3.3 V DC
3	COM	Black	Ground
4	5 V	Red	+5 V DC
5	COM	Black	Ground
6	5 V	Red	+5 V DC
7	COM	Black	Ground
8	PWR_OK	Grey	Status signal generated to notify that the operating voltages of the source are within the margins necessary for its correct operation. +5 V DC if everything is ok
9	5 VSB	Purple	Standby voltage (max. 10 mA)
10	12 V	Yellow	+12 V DC
11	12 V	Yellow	+12 V DC
12	3.3 V	Orange	3.3 V DC
13	3.3 V	Orange	3.3 V DC
14	-12 V	Blue	-12 V DC
15	COM	Black	Ground
16	PS_ON	Green	This PIN must be connected to ground to start the power supply
17	COM	Black	Ground
18	COM	Black	Ground
19	COM	Black	Ground
20	n.c.	White	n.c.
21	5 V	Red	+5 V DC
22	5 V	Red	+5 V DC
23	5 V	Red	+5 V DC
24	COM	Black	Ground

Table 4.1 – *ATX PIN description [29]*

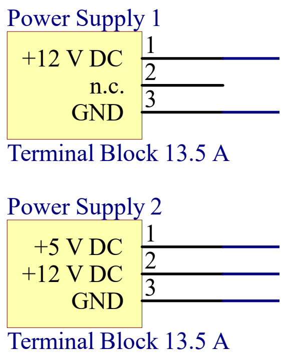
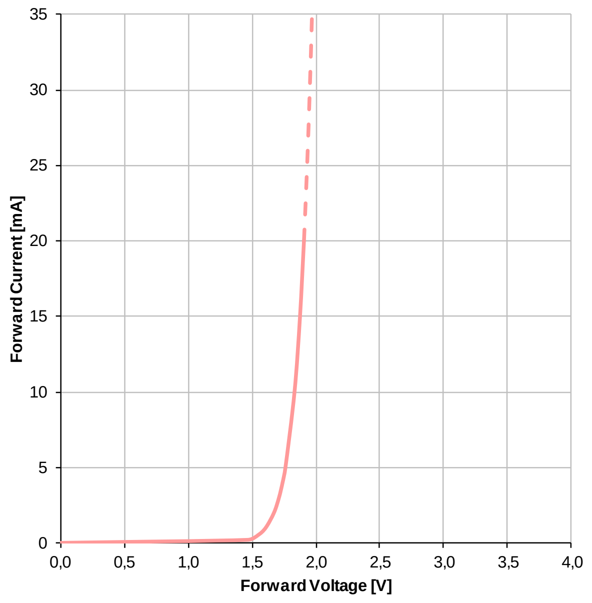
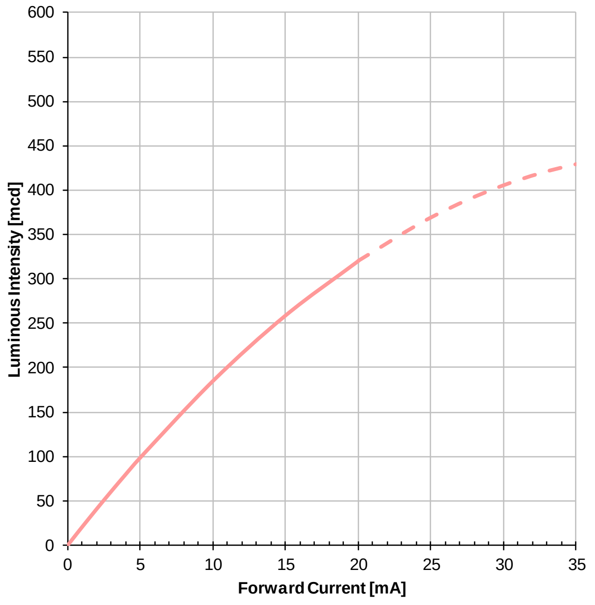


Figure 4.8 – *ATX switched power supply schematic symbol*

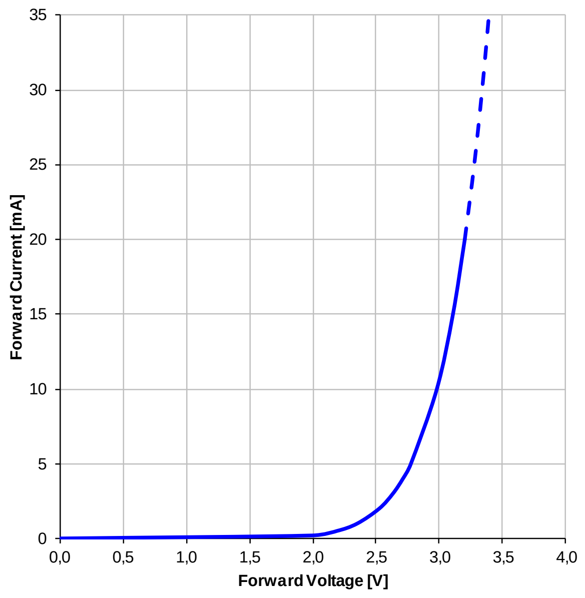


(a) *Red LED I-V characteristic*

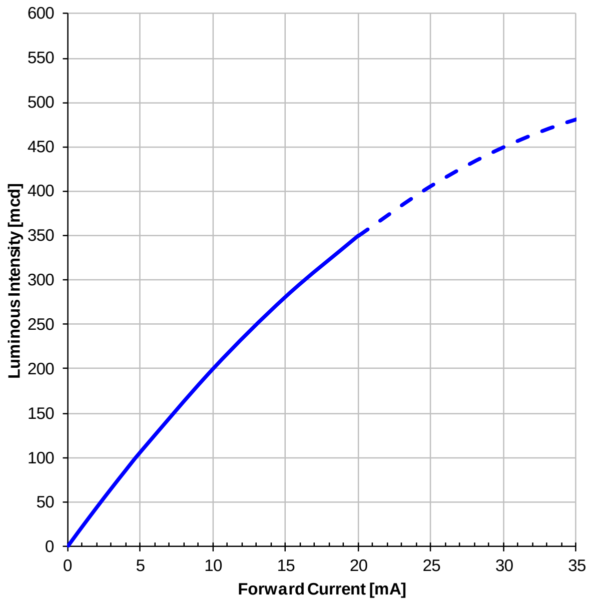


(b) *Red LED luminous intensity curve*

Figure 4.9 – *Characteristic curves of the red LED [35]*



(a) *Blue LED I-V characteristic*



(b) *Blue LED luminous intensity curve*

Figure 4.10 – *Characteristic curves of the blue LED [34]*

In Figure 4.9 and Figure 4.10 we can see some characteristic curves of both LEDs. The curve on the left indicates the current flowing through the LED as a function of the voltage difference between its terminals. We can see that, at a current of 30 mA, about 3.3 V drops in the blue LED while only 2 V drops in the red one, so the blue LED dissipates more power than the red one.

On the other hand, for a current of 30 mA, the red LED emits a luminous intensity of 400 mcd, while the blue one emits 450 mcd.

We must add a resistor in series with each LED to ensure that about 30 mA circulate through both, making them work at a similar intensity so that neither shines much brighter than the other. To calculate the value of this resistor we simply have to perform the following calculation:

$$R_{LED-RED} = \frac{12V - 2V}{30mA} = 333.33\ \Omega \approx 330\ \Omega \tag{4.1.1}$$

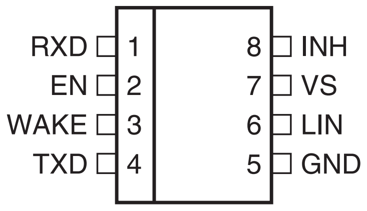
$$R_{LED-BLUE} = \frac{5V - 3.3V}{30mA} = 56.67\ \Omega \approx 62\ \Omega \tag{4.1.2}$$

4.1.1.3 LIN transceiver module

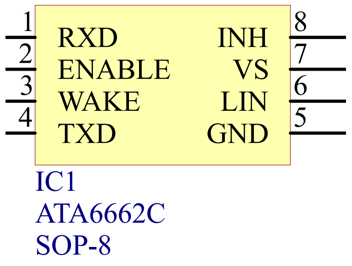
This module is used to convert the serial communication provided by the Arduino or the PC to LIN communication, necessary for the control of the pilots. Remember that, in subsection 3.2.3, we chose the **ATA6662C** as the main component to perform this function.

The Figure 4.11 (a) shows the pinout corresponding to the **SOP-8** encapsulation of the **ATA6662C**.

In Figure 4.11 (b) we can see the symbol created by us in Altium Designer® 21 and manually added to our library that will represent the **ATA6662C** in the schematic.



(a) *ATA6662C pinout [2]*



(b) *ATA6662C schematic symbol*

Figure 4.11 – *ATA6662C pinout and schematic symbol*

The Table 4.2 extracted of the ATA6662C official datasheet shows the description of the different functions that each PIN of the device performs

PIN	SYMBOL	FUNCTION
1	RXD	Receive Data Output (OD)
2	EN	Enables Normal Mode; when the input is open or low, the device is in Sleep Mode
3	WAKE	High voltage input for local wake-up request. If not needed, connect directly to VS
4	TXD	Transmit data input; active low output (strong pull-down) after a local wake-up request
5	GND	Ground, heat sink
6	LIN	LIN bus line input/output
7	VS	Battery Supply
8	INH	Battery-related inhibit output for controlling an external voltage regulator; active high after a wake-up request

Table 4.2 – *ATA6662C PIN description [2]*

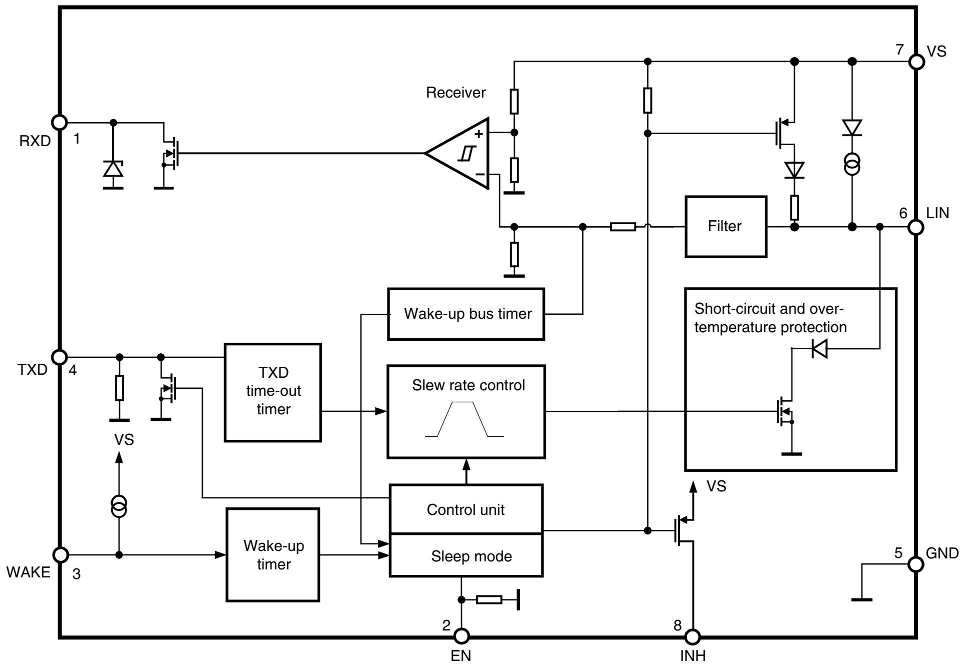


Figure 4.12 – *ATA6662C block diagram [2]*

The Figure 4.12 shows the internal block level diagram of the ATA6662C showing a summary of the IC's operation.

Next, we are going to carry out an analysis of the nature of each IC PIN and draw conclusions about which components we must connect to this module and how we must do it to condition its operation.

- **RXD (output PIN):** This PIN reports to the microcontroller Arduino UNO or the CH340 the state of the LIN bus. LIN high (recessive) is reported by a high level at RXD, LIN low (dominant) is reported by a low voltage at RXD.
  - It must be connected to the PIN TX of the Arduino UNO or the CH340. This is because, in a serial communication like this, one of the modules sends data that the other receives and vice versa. That is why the two cables that make up the connection must be connected crossed.
  - Is in an **open collector** [1] configuration, so we'll need to somehow **implement a pull-up** on this PIN. For this case we have chosen the recommended pull-up implementation present in the datasheet [2], which consists of the parallel connection of the following components:
    - **20 pF capacitor**
    - **1 KΩ resistor**
  - It would be a good idea to add an LED that visually confirms that data is being transmitted through to the Arduino UNO or CH340. In this case we have added a blue 0603 LED whose characteristic are showed in Figure 4.10. Being in the same conditions of use as in subsubsection 4.1.1.2 we will use the same resistor value: 62 Ω
- **TXD (I/O PIN):** is the microcontroller interface to control the state of the LIN output.
  - It must be connected to the PIN RX of the Arduino UNO or the CH340 for the same reason explained in the previous point.

[1]Open Collector: is a common type of output found on many ICs, which behaves like a switch that is either connected to ground or disconnected. Instead of outputting a signal of a specific voltage or current, the output signal is applied to the base of an internal NPN transistor whose collector is externalized (open) on a PIN of the IC.

- As in the previous case, it would be a good idea to add a LED that visually confirms that data is being transmitted to / from the Arduino UNO or CH340.

- **WAKE (Input PIN)**: This PIN is a **high-voltage input** used to **wake the device up from Sleep Mode**. We don't need a local wake-up in our application, so we will **connect PIN WAKE directly to PIN VS.**

- **INH (Inhibit Output) PIN:** This PIN is used to control an external switchable voltage regulator having a wake-up input. As we will not use one of them, we will leave it without connecting.

- **LIN (Bus PIN):** A **low-side driver** with **internal current limitation** and **thermal shutdown** and an **internal pull-up resistor** are implemented as specified for LIN 2.x. The voltage range is **from −27V to +40V**
  - This PIN must be connected to the LIN bus where the LIN slaves (pilots) are connected which os accessible through the corresponding terminal block.
  - It is mandatory to include a **pull-up** if we are going to use this IC as the master node of the LIN communication (which is our case). So we will add a 1 KΩ resistor as specified in the datasheet [2].
  - As in the previous case, it would be a good idea to add an LED that visually confirms that data is being transmitted to / from the LIN pilots. In this case we have added a blue 0603 LED, whose series resistance we calculate below:

$$R_{LED-BLUE} = \frac{12V - 3.3V}{30mA} = 290\ \Omega \approx 300\ \Omega \quad (4.1.3)$$

  - We will add a capacitor to GND to form a **low-pass filter** for the LIN bus, as it removes high-frequency signals from the bus by giving those signals a low-impedance path to GND.

- **EN PIN (Enable Input):** This pin controls the **Operation Mode** of the interface. We must connect it to an I/O pin of the microcontroller Arduino UNO

- **PIN VS (Supply PIN): Undervoltage detection** is implemented to disable transmission if VS falls to a value below 5 V in order to **avoid false bus messages**. After switching on VS, the IC switches to Fail-safe Mode and INHIBIT is switched on. The supply current in Sleep Mode is typically 10 uA.
  - It is recommended to include some **decoupling capacitors** between VS and GND that act as a **low-pass filter for high frequency noise**. For this application we will use the values recommended in the ATA6662C Datasheet:
    * **C1 = 0.1 uF**
    * **C2 = 22 uF**
  - We will include a **1N4148 diode** between the +12 V DC source voltage and the VS PIN as **reverse current protection.** The choice of this diode has been based on its availability in the laboratory.

- **GND PIN (Ground PIN)**: The ATA6662C does not affect the LIN Bus in the case of a GND disconnection. It is able to **handle a ground shift up** to 11.5% of VS. We will simply connect this PIN **to the common ground** of our schematic.

In Figure 4.13 we can see how the module would turn out after attending to all the conclusions drawn for each PIN of the

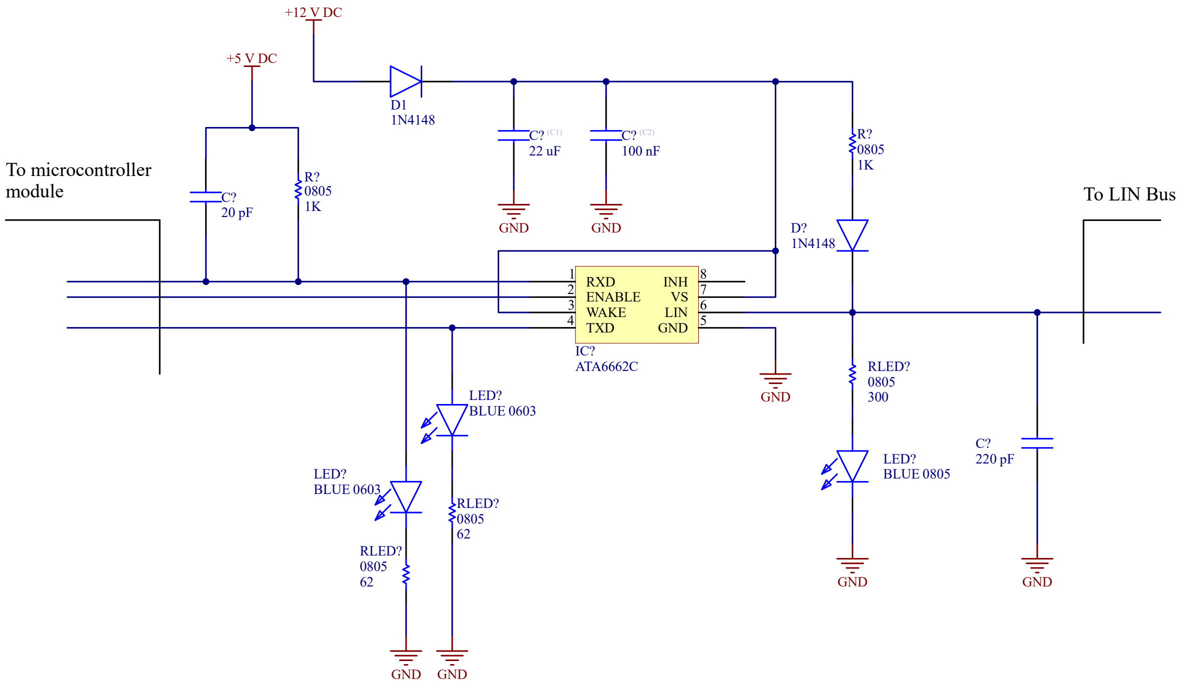
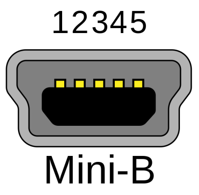


Figure 4.13 – *ATA6662C aplication circuit*

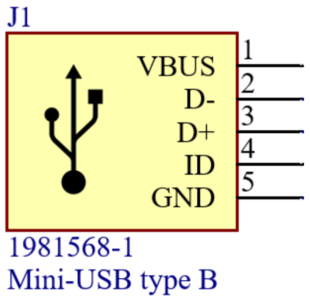
4.1.1.4 USB port module

This is the module in charge of implementing the connection between our product and the PC. In subsection 3.2.4, we decided to use a **mini-USB type B** port for this module.

In Figure C.3 (a) we can see the pinout the mini-USB type B connector, which PIN description is explained in the Table 4.3.



(a) *Mini-USB type B pinout*



(b) *Mini-USB type B schematic symbol*

Figure 4.14 – *Mini-USB type B pinout and schematic symbol*

PIN	SYMBOL	Cable Color	Function
1	VBUS	Red	+5 V DC
2	D-	White	Data -
3	D+	Green	Data +
4	ID	N/A	Permits distinction from device connection: - Host: connected to the signal ground. - Device: not connected
5	GND	Black	Signal ground

Table 4.3 – *Mini-USB type B pin description*

In Figure C.3 (b) we can see the schematic symbol that we manually created in Altium Designer® 21 and added to our customized library in order to include it in our project schematic.

Next, as in the previous section, we will draw conclusions about where we should connect each PIN of the device for the correct performance of all the modules.

- **VBUS (Supply PIN):** Since our PCB won't receive the +5 V DC power supply from the USB connection, we will leave this PIN unconnected.
- **D- (Data - PIN)**: We need to connect this PIN to the corresponding D- PIN in the ***USB to Serial*** module, as we want to convert the USB communication to serial communication.
- **D+ (Data + PIN)**: We need to connect this PIN to the corresponding D+ PIN in the ***USB to Serial*** module, as we want to convert the USB communication to serial communication.
- **ID (Identifier PIN):** We will leave this PIN unconnected, since our product will not be the host in the USB communication.
- **GND (Ground PIN):** We will connect this pin to the general GND of our project.

Finally, with all this considerations, the USB port module connections results in the application circuit that we can see in Figure 4.15.

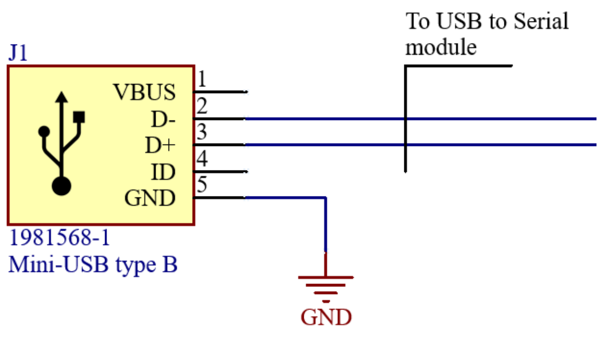


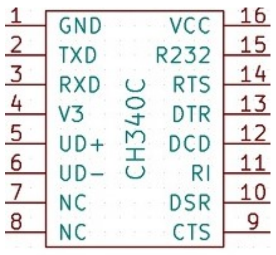
Figure 4.15 – *Mini-USB type B module application circuit*

4.1.1.5 USB to serial module

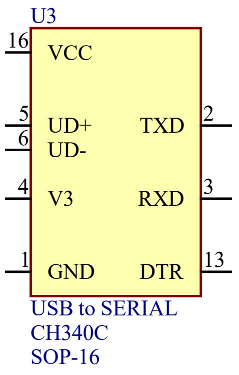
In subsection 3.2.5 we decided to use the **CH340C** to perform the function of USB to serial converter. This module will be in charge of establishing a connection interface **between USB communication and**

serial communication. It is necessary to install the corresponding drivers on the PC so that the CH340C is correctly detected by the PC as a serial communications port. The steps to follow are in section C.2

In Figure 4.16 (a) we can see the pinout the CH340C IC, which PIN description is explained in the Table 4.4, showing in bold the pins' symbols that we are going to use for the correct operation of our desired application.



(a) *CH340C IC pinout* [47]



(b) *CH340C Altium Designer® 21 schematic symbol*

Figure 4.16 – *CH340C pinout and schematic symbol*

PIN	SYMBOL	TYPE	FUNCTION
1	**GND**	POWER	Ground connection for USB
2	**TXD**	OUT	Serial data output
3	**RXD**	IN	Serial data input, set up controlled pull-up and pull-down resistor
4	**V3**	POWER	- Connects of VCC to input outside power while 3.3V. - Connects of 0.01uF decoupling capacitance outside while 5V
5	**UD+**	USB Signal	Directly connect to D+ data wire of USB bus, set up pull-up resistor internal
6	**UD-**	USB Signal	Directly connect to D- data wire of USB bus
7	N.C.	N/A	N/A
8	N.C.	N/A	N/A
9	CTS	IN	MODEM liaison input signal, clear sending, active with low
10	DSR	IN	MODEM liaison input signal, data equipment is ready, active with low
11	RI	IN	MODEM liaison input signal, oscillate ring to prompt, active with low
12	DCD	OUT	MODEM liaison input signal, carrier wave detection, active with low
13	**DTR**	OUT	MODEM liaison output signal, data endpoint is ready, active with low
14	RTS	OUT	MODEM liaison output signal, request to send, active with low
15	R232	IN	Assistant RS232 enable, active with high, set up pull-down resistor internal
16	**VCC**	POWER	Positive power input port, requires an external 0.1uF power decoupling capacitance

Table 4.4 – *CH340C PIN description* [47]

In Figure 4.16 (b) we can see the schematic symbol created manually in Altium Designer® 21 and added to our customized component schematic library in order to get the component ready to place in the schematic of the project.

As we can see, there are a lot of pins in the CH340C (SOP-16 package) that we are **not going to use** for our application, so we have **deleted them in its schematic symbol** in order to offer a **cleaner and clearer** view in the final schematic. Anyways, the information about the function that every PIN of the

CH340C does is available in Table 4.4.

Next, as in previous sections, we will analyze the requirements of each PIN used and justify its connection with the different components and with the other modules.

- **VCC (Power PIN):** As specified in the IC Datasheet, we will power the CH340C with **+5 V DC**. As before, we are going to include **two decoupling capacitors** as a low-pass-filter to eliminate high frequency noise. In this case their capacitance (as recommended in CH340C Datasheet) are:

  - $C_5 = 0.1$ uF
  - $C_6 = 10$ uF

- **UD- (USB Signal PIN):** We must connect this PIN to the corresponding D- PIN of the mini-USB type B module.

- **UD+ (USB Signal PIN):** We must connect this PIN to the corresponding D+ PIN of the mini-USB type B module.

- **V3 (Power PIN):** Since we are going to power the CH340C with +5 V DC, we must connect this PIN to ground through **C7 = 0.01 uF** Bypass Capacitor as specified in the CH340C Datasheet.

- **TXD (Serial data output PIN):** This PIN must be connected to the corresponding RXD PIN of the LIN transceiver module.

  Additionally, we decided to include a *jumper* in order to avoid communication collision when Arduino UNO is chosen to send frames to the LIN transceiver module.
  Furthermore, it will also be connected to the corresponding RX PIN in the microcontroller module (Arduino UNO) to be able to program it externally.

- **RXD (Serial data input PIN):** This PIN must be connected to the corresponding TXD PIN of the LIN transceiver module.
  Furthermore, it will also be connected to the corresponding TX PIN in the microcontroller module to be able to program it externally.

- **DTR (Reset PIN):** We must connect this PIN to the corresponding RESET PIN in the microcontrolller module so that the Arduino UNO resets automatically when it finishes programming.

- **GND (Power PIN):** We must connect this PIN to the common ground PIN of our entire schematic.

In Figure 4.17 we can see the application circuit for this module according with the conclusions obtained for each PIN.

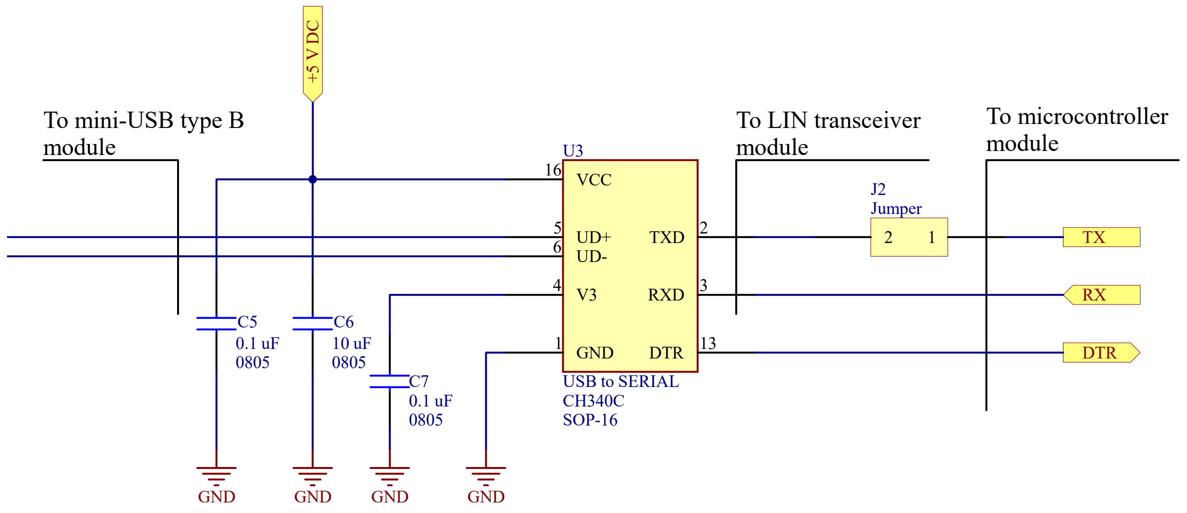


Figure 4.17 – *CH340C Altium Designer® 21 application circuit schematic*

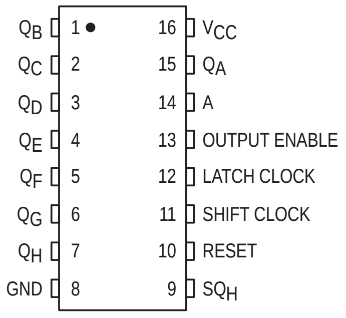
4.1.1.6 Shift register module

In subsection 3.2.6 we decided to use some **74HC595** ICs to increase the number of digital outputs of the Arduino UNO. We needed it because we need to control each function of the ON/OFF pilots separately, which requires several more digital outputs than Arduino UNO has.

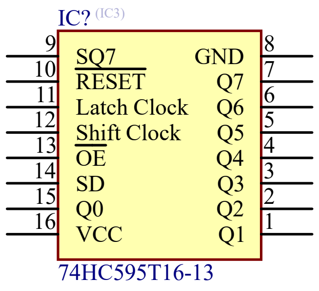
	SN74HC595
MANUFACTURER	ON-Semiconductor
BITS OUTPUT	8
PACKAGE	TSSOP-16
TEMPERATURE RANGE	-55 to 125 ºC
CASCADE	YES
PRICE	0.09 € (es.aliexpress.com)

Table 4.5 – *74HC595 main characteristics*

In Figure 4.18 (a) we can see the 74HC595 pinout, which description is shown in Table 4.6.



(a) *74HC595 pinout*



(b) *74HC595 Altium Designer® 21 schematic symbol*

Figure 4.18 – *74HC595 pinout and schematic symbol*

In Figure 4.19 and Figure 4.20 (expanded) we can see the logic diagram of the 74HC595, showing the internal high-level workings of the IC. We can see that **two different clock signals are used**, one that marks the rate at which the bits in the shift register are shifted (shift clock) and another that marks the rate at which the bits in the shift register move to the output (latch clock). That is why, in order to control these two events separately, we must connect these clocks to different PINs of the Arduino UNO

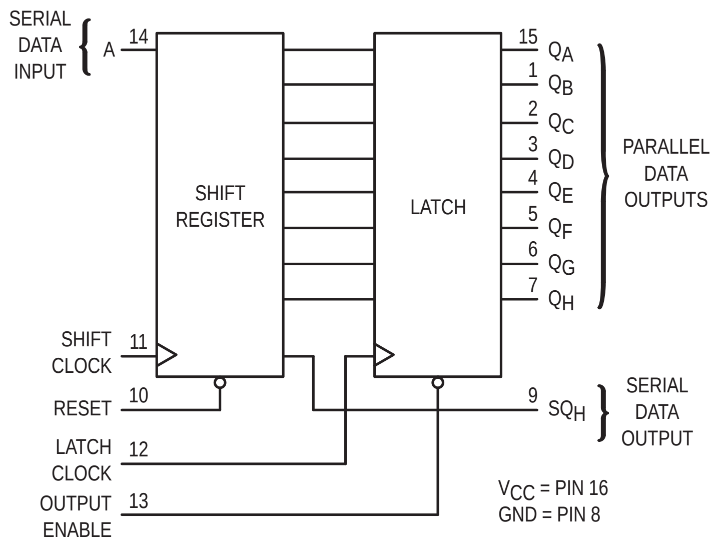


Figure 4.19 – *74HC595 logic diagram [42]*

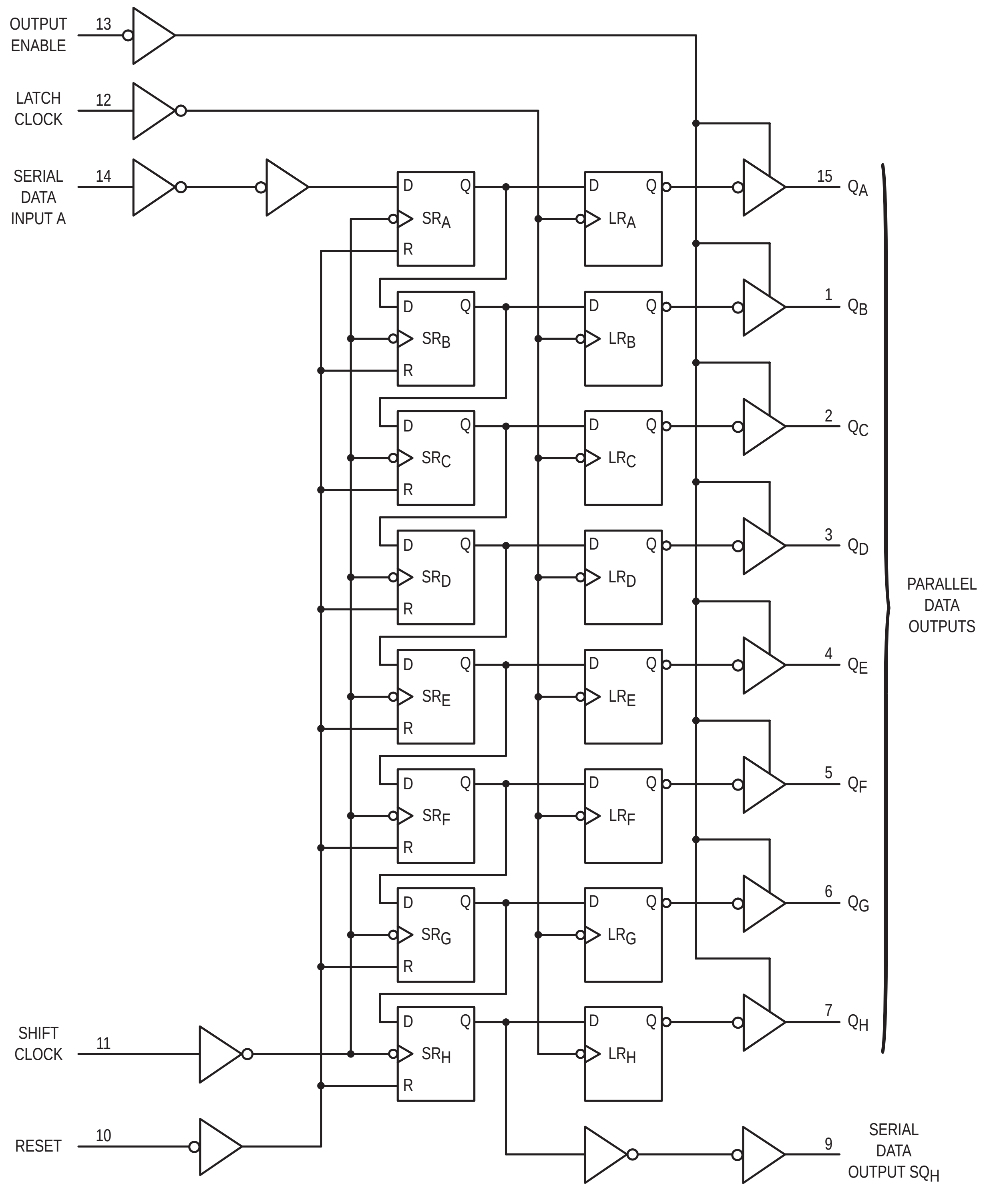


Figure 4.20 – *74HC595 logic diagram (expanded) [42]*

PIN(s)	SYMBOL	TYPE	FUNCTION
1	A	INPUT	Serial Data Input. The data on this pin is shifted into the 8–bit serial shift register.
15, 1 - 7	Q_A - Q_H	OUT	Noninverted, 3–state, latch outputs
9	SQ_H	OUTPUT	Noninverted, Serial Data Output. This is the output of the eighth stage of the 8–bit shift register. This output does not have three–state capability.
10	RESET	INPUT	DActive–low, Asynchronous, Shift Register Reset Input. A low on this pin resets the shift register portion of this device only. The 8–bit latch is not affected.
11	Shift Clock	INPUT	Shift Register Clock Input. A low– to–high transition on this input causes the data at the Serial Input pin to be shifted into the 8–bit shift register.
12	Latch Clock	INPUT	Storage Latch Clock Input. A low–to–high transition on this input latches the shift register data.
13	Output Enable	INPUT	Active–low Output Enable. A low on this input allows the data from the latches to be presented at the outputs. A high on this input forces the outputs (QA–QH) into the high–impedance state. The serial output is not affected by this control unit.
8	GND	POWER	Ground
16	VCC	POWER	+5 V DC

Table 4.6 – *74HC595 PIN description* [42]

Next, as in previous sections, from the description of the functions of each PIN specified in Table 4.6 , we will proceed to draw conclusions about which components we must connect to each PIN of these ICs, how we must connect them to each other and to the rest of the modules. for its correct operation.

Something important to keep in mind before starting with the design of the connection of this module is that, as we said in subsection 3.2.6, we will need to cascade a total of **4 units of 74HC595** to provide the 32 digital outputs necessary for the control of the ON/OFF pilots. This is why in the following list, the conclusions drawn for each PIN will refer to the 4 ICs at the same time, specifying the exceptions in each case.

- **A (Data serial input)**: In the first of the 4 ICs, this PIN must be connected to any digital PIN of the Arduino UNO, since this will provide the information in serial communication format that the ICs must convert to parallel output.

  For the successive ICs in cascade, this PIN must be **connected to the SQ_H PIN** of the immediately previous IC, since this PIN is the one that provides the serial information on the **status of the eighth bit** of the shift register, transferring this information to the serial input of the following IC.

- Q_A **-** Q_H **(Parallel data output)**: These pins correspond to the 32 digital outputs of the module, so they must be connected directly to the **power amplifier module.**

- SQ_H **(Serial data output)**: As we said, this must always be connected to pin A of the next IC, except in the last one of the cascade, where we will leave it unconnected.

- **RESET**: Since we are not too interested in implementing a RESET function in the shift registers, as it is an active-low PIN, we will connect it directly to +5 V DC in all the ICs in the cascade.

- **Shift Clock**: We will connect this PIN to the same digital output of the Arduino UNO in all the ICs of the cascade, since we must generate the same clock so that all the ICs work in a coordinated manner as if it were a single IC with a 32-bit parallel output.

- **Latch Clock**: Idem. the previous.

- **Output Enable**: We will connect this PIN to the same digital output of the Arduino UNO to be able to enable/disable the output of all shift registers at the same time.

- **GND (Ground PIN)**: This PIN must be connected to the common ground of all our schematic in all cases.

- **VCC (Supply PIN)**: we will connect this PIN to the +5 V DC supply on all ICs

In Figure 4.18 (b) the schematic symbol created in Altium Designer® 21 and added to our customized library which we are going to use in the general schematic is shown.

In Figure 4.21 is shown the final application circuit which shows how the 4 ICs make up the shift register module attending to the previous PIN description.

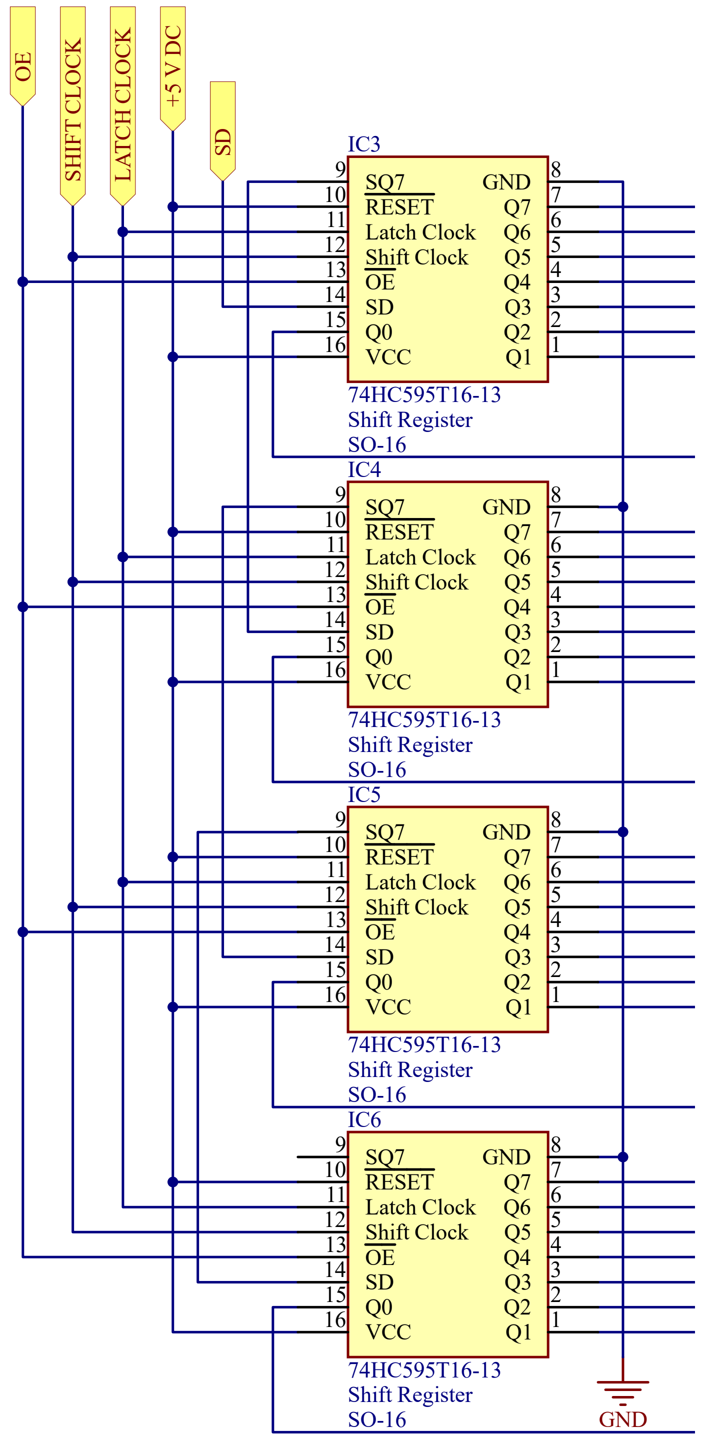


Figure 4.21 – *74HC595 application circuit*

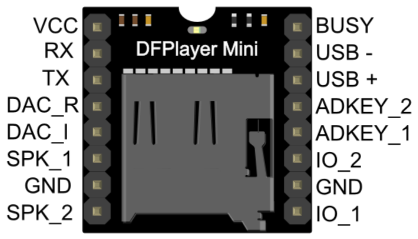
4.1.1.7 Audio player module

In subsection 3.2.7 we decided to use a DF-Player mini to implement the mp3 player module. This device will take care of **playing audio files** through a speaker, since it also incorporates an **internal audio amplifier**. Although the device has the ability to play audio directly from a microSD card or from USB communication, we have decided that audio samples are going to be generated by the Arduino UNO, which will communicate with the DF-Player mini through a VSP (using two digital pins).

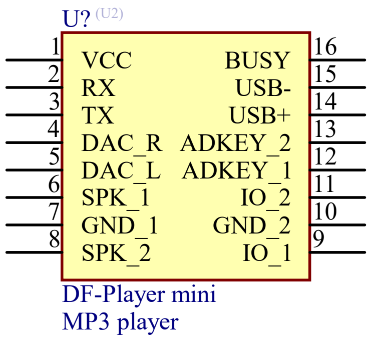
	DF Player Mini
MANUFACTURER	Arduino
DIMENSIONS	20 x 20 mm
VOLTAGE RANGE	3.2 - 5 V
CURRENT	20 mA
TEMPERATURE RANGE	-40 to 70 ºC
PRICE	0.97 € (es.aliexpress.com)

Table 4.7 – *DF-Player mini main characteristics*

In Figure 4.22 (a) we can see the pinout of the DF-Player mini, which PIN description is shown in Table 4.8. In the case of this module we also have a device with many more pins than we are going to need for our application, so we will show the ones that are useful to us in bold.



(a) *DF-Player mini pinout*



(b) *DF-Player mini Altium Designer® 21 schematic symbol*

Figure 4.22 – *DF-Player mini pinout and schematic symbol*

PIN	SYMBOL	TYPE	FUNCTION
1	**VCC**	POWER	3.3 to 5 V voltage supply.
2	**RX**	INPUT	Microcontroller serial input
3	**TX**	OUTPUT	Microcontroller serial output
4	DAC_R	OUTPUT	Audio output right channel, drive earphone and amplifier
5	DAC_L	OUTPUT	Audio output left channel, drive earphone and amplifier
6	**SPK2**	OUTPUT	Drive speaker less than 3W
7	**GND**	POWER	Power ground
8	**SPK1**	OUTPUT	Drive speaker less than 3W
9	IO1	TRIGGER	Short press to play previous(long press to decrease volume)
10	**GND**	POWER	Power ground
11	IO2	TRIGGER	Short pree to play next (long press to increase volume)
12	ADKEY1	TRIGGER	Trigger play first segment
12	ADKEY2	TRIGGER	Trigger play fifth segment
14	USB+	INPUT	USB port positive
15	USB-	INPUT	USB port negative
16	Busy	OUTPUT	Low means playing High means no

Table 4.8 – *DF-Player mini PIN description*

In Figure 4.22 (b) the schematic symbol created in Altium Designer® 21 and added to our customized library which we are going to use in the general schematic is shown.

In Figure 4.23 we can see the application circuit for this module according with the conclusions obtained for each PIN.

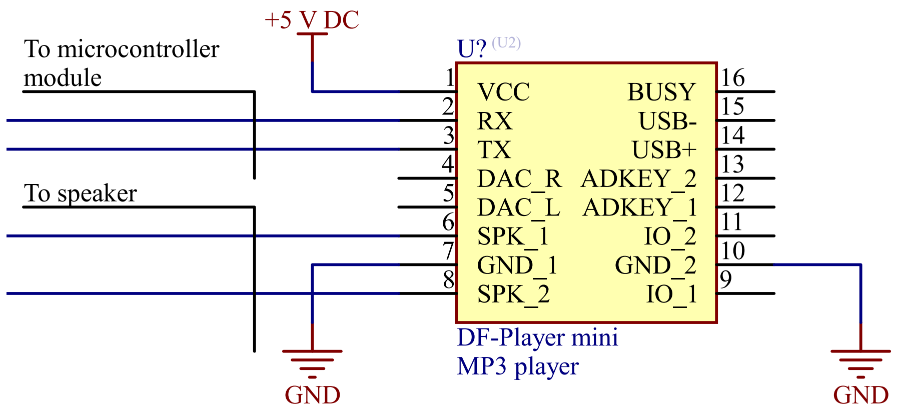


Figure 4.23 – *DF-Player mini application circuit*

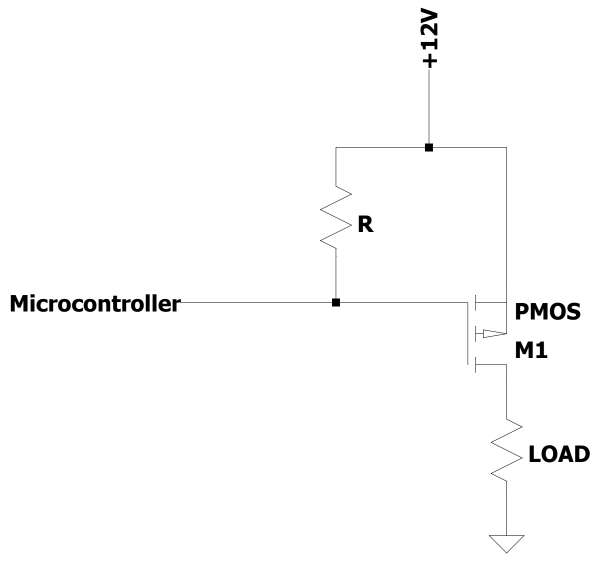
4.1.1.8 Power amplifier module

In subsection 3.2.8 we realized that we would need to include a **power amplifier module** in our design. Since the power needed to activate the functions of the pilots is much higher than the maximum power that the digital pins of the Arduino UNO are capable of providing.

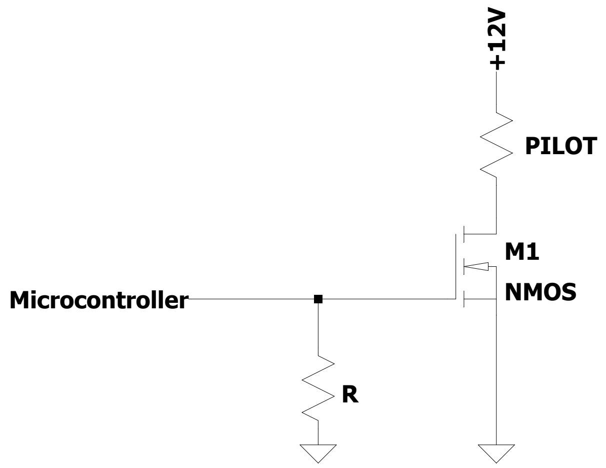
Two different topologies have been considered to implement this power amplifier:

- **Low side transistor switch (Figure 4.24 (a)):** It is built using a **P-channel MOSFET transistor**, the load is placed in the drain of the transistor, being always connected to ground and receiving +12 V DC when the transistor is closed, thus closing the circuit.
- **High side transistor switch (Figure 4.24 (b)):** It is built using an **N-channel MOSFET transistor**, the load is placed on the drain of the transistor, being always connected to +12 V DC and connecting to ground when the MOSFET is closed, thus closing the circuit.

Both options include a **pull-up** (in the case of the low-side transistor switch) or **pull-down** (in the case of the high-side transistor switch) **resistor**. This is done because it is possible that, when a RESET occurs in the microcontroller, the digital pins that control the gates of the MOSFETs remain in a "floating" state, leaving the circuit open. These resistors **close the circuit during those moments.**



(a) *Low side transistor switch schematic*



(b) *High side transistor switch schematic*

Figure 4.24 – *Implementation alternatives considered for the power amplifier*

Between these two possibilities, it is necessary to use the **low-side transistor switch**. This is because all the different functions inside an ON/OFF pilot **share the same ground connection**, that is, we can only control if they are connected to +12 V DC or not. Therefore, this makes it impossible to use high-side transistor switch and **only the first option works.**

Once this choice is made, another problem arises. Let us recall the expressions that govern the different conduction states of a MOSFET transistor.

STATE	VOLTAGE CONDITION
CUT OFF	$V_{SG} <\mid V_T \mid$
LINEAR	$V_{SG} >\mid V_T \mid$ $V_{SD} < V_{SG} - \mid V_T \mid$
SATURATION	$V_{SG} > V_T$ $V_{SD} > V_{SG} - \mid V_T \mid$

Table 4.9 – *P-channel MOSFET transistor state voltage conditions*

For the desired application, the P-channel transistor is required to operate between **cutoff and saturation modes** since it is necessary that the specific function of the pilot be completely turned off or on.

As we can see in Table 4.9, for the transistor to operate in saturation mode (the one necessary for there to be no current limitation), the voltage **between the source and the drain** must exceed the voltage **between the source and the gate**, subtracting the threshold voltage.

Similarly, for the P-channel transistor to cut-off, the voltage between the source and the gate must be **less than the threshold voltage.**. This is where the problem of using the circuit described in Figure 4.24 (a) arises. It is known that the high voltage used by the digital I/O pins of the Arduino UNO is **+5 V DC**, which is V_G. As the schematic of the low-side transistor switch shows, the source of the P-channel transistor must be connected to the **+12 V DC** supply (V_S) to power the pilot when the transistor is closed. Therefore, the voltage between the source and the gate when the digital output is high (V_{SG}) will be:

$$V_{SG} = V_S - V_G = 12V - 5V = 7V \ > \mid V_{TP} \mid \approx \ [0.7 - 1V] \tag{4.1.4}$$

With which, it is concluded that it is **impossible to turn off the P-channel transistor** with only +5 V DC at its gate. At least a voltage equal to the supply voltage would be needed, **that is,** V_G **= +12 V DC**

To solve this problem, it has been decided to use a **second transistor** to control the first. That is, it will be a transistor driving a transistor. In this way, it will be possible to get enough voltage at the gate of the P-channel transistor to be able to turn it off. For this second transistor, it has been chosen to make use of an **N-channel MOSFET transistor**, since another P-channel would make the problem remain. The final configuration of this power amplifier module including the second MOSFET would be as shown in 4.25

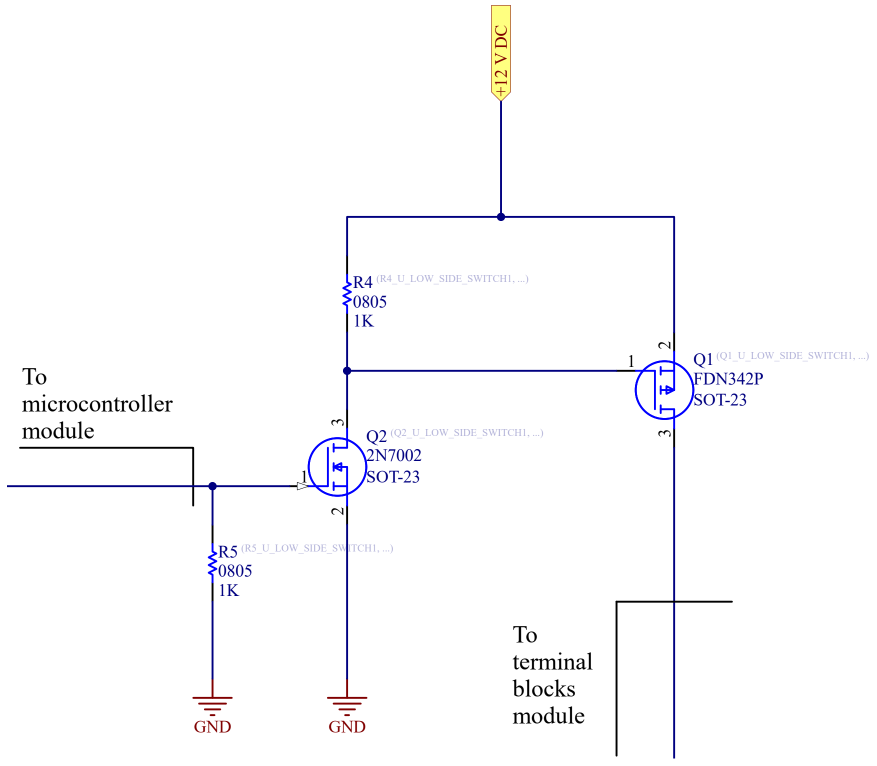

+12 V DC
R4 (R4_U_LOW_SIDE_SWITCH1, ...)
0805
1K
Q1 (Q1_U_LOW_SIDE_SWITCH1, ...)
FDN342P
SOT-23
Q2 (Q2_U_LOW_SIDE_SWITCH1, ...)
2N7002
SOT-23
R5 (R5_U_LOW_SIDE_SWITCH1, ...)
0805
1K
To microcontroller module
To terminal blocks module
GND
GND

Figure 4.25 – *Power amplifier module implemented circuit*

As you can see, a resistor R4 has been included in the drain of the N-channel MOSFET transistor. If this resistance is not included, the voltage at the drain of the N-channel MOSFET will always be +12 V DC and, therefore, it will also be the same at the gate of the P-channel MOSFET, which would not allow the latter to enter in cut off. The value of this resistor is not critical.

By implementing this circuit, we get the signal present at the gate of the P-channel MOSFET transistor (the one that controls its opening and closing) to vary between 0 and +12 V DC. This ensures that this transistor can be turned off without problem.

For the implementation of this module it has been chosen to use the **FDN342P** as **P-channel MOSFET** transistor and **2N7002** as N-channel MOSFET transistor. Their main characteristics are shown in Table 4.10.

	FDN342P	**2N7002**
CHANNEL	P	N
MANUFACTURER	Fairchild semiconductor	KD electronics
PACKAGE	SOT-23	SOT-23
S-D (D-S) MAX. DC VOLTAGE	20 V	60 V
MAX. DC CURRENT	2 A	115 mA
TEMPERATURE RANGE	-55 to 150 ºC	-55 to 150 ºC
PRICE	0.02 € (es.aliexpress.com)	0.01 € (es.aliexpress.com)

Table 4.10 – *FDN342P and 2N7002 main characteristics*

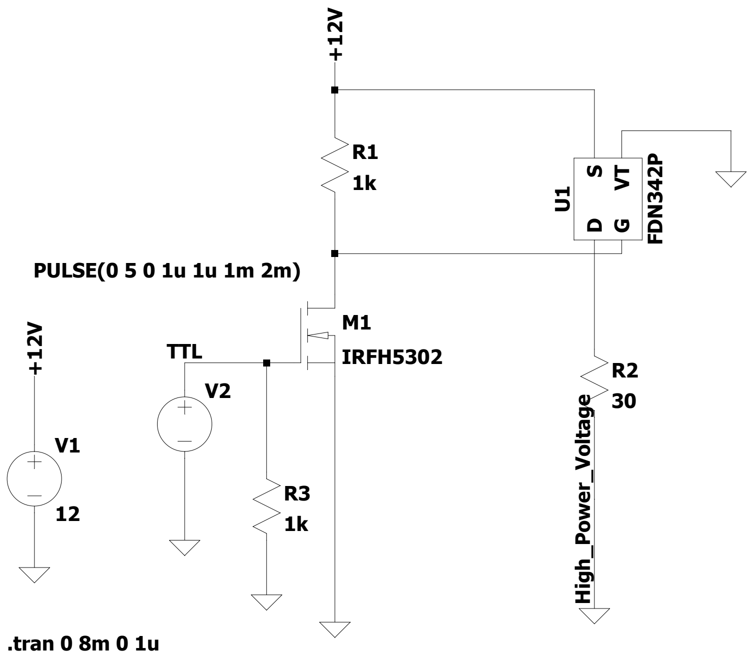
For this case, a simulation of this circuit has been carried out in the software SPICE to check that it really behaves as we expect. To do this, we first had to find a ***.SUBCKT*** file corresponding to the FDN342P transistor that models, based on certain parameters, the real behavior of the component.

Figure 4.26 (a) shows the circuit that has had to be implemented in SPICE in order to simulate its operation. Let us remember that the corresponding digital PIN of the Arduino UNO offers an output of +5 V when it is high and 0 V for low. That is why, to carry out the test, we have introduced a square signal through the gate of the N-channel transistor 2N7002 whose voltage levels vary between 0 and +5 V DC in order to be able to clearly observe the behavior at the output.

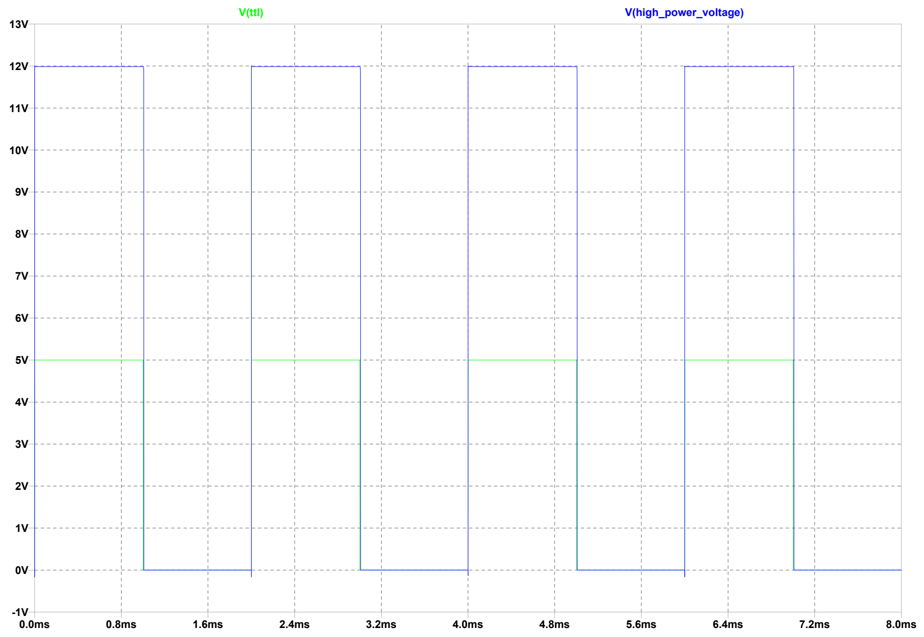
As load resistance we have used a value of 30 KΩ, which means that, when the P-channel transistor FDN342P is operating in saturation mode (see Table 4.9), a current of about **400 mA** circulates through it, which is realistic with the approximate currents required. many of the functions of the different headlights.

The VT PIN present in the circuit model of the p-channel transistor FDN342P serves to establish a fictitious temperature at which the transistor would be in operating conditions. We have decided to connect this PIN to ground so that the temperature taken into account in the simulations is the normal room temperature of 25ºC.

Finally, carrying out a transient type simulation (which represents the voltage/current at the chosen point with respect to time) we obtain what is seen in Figure 4.26 (b). We can see that when there is +5V at the gate of the N-channel transistor, this causes it to go into saturation mode and short the transistor gate of the P-channel transistor to ground, which in turn causes the P-channel transistor to go into saturation mode and pass all the current from the +12 V power supply to the load present on its drain, which represents the function of the headlight in question. We conclude that it is a non-inverting amplifier because when we apply voltage to the input, the output is activated.



(a) *Power amplifier module implemented circuit in SPICE*



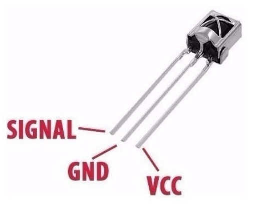
(b) *Power amplifier module behaviour simulation*

Figure 4.26 – *Simulation of power amplifier module in SPICE software*

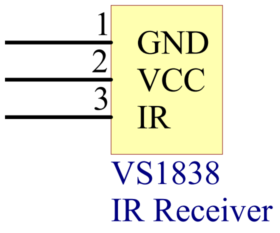
4.1.1.9 Remote control module

In subsection 3.2.9, we decided that the method of remote control of the device that best suited our needs was an IR receiver. Specifically, we decided to use the **VS1838**, an IR receiver fully compatible with

Arduino UNO and easy to program. Table 4.11 shows the main characteristics of this component.



(a) *Pinout of the VS1838*



(b) *High side transistor switch schematic*

Figure 4.27 – *VS1838 pinout and schematic symbol*

	VS1838
MANUFACTURER	ETC2
VOLTAGE RANGE	Min.: 2.7 V Typ.: 5 V Max.: 5.5 V
CURRENT CONSUMPTION RANGE	Typ.: 0.4 mA Max.: 1.5 mA
TEMPERATURE RANGE	-40 to 125 ºC
PRICE	0.05 € (es.aliexpress.com)

Table 4.11 – *VS1838 main characteristics [19]*

The connection scheme of this component is very simple. We simply have to feed it through its power pins and connect the data PIN to any digital pin of the Arduino UNO. Table 4.12 shows the VS1838 PIN description.

PIN(s)	**SYMBOL**	**TYPE**	**FUNCTION**
1	IR	OUTPUT	Gives the sequence to the microcontroller depending on the IR signal that has been detected
2	GND	POWER	Ground
3	VCC	POWER	Power supply PIN. Can be supplied with 2.5 to 5 V voltage.

Table 4.12 – *VS1838 PIN description [19]*

Figure 4.27 (b) shows the schematic symbol manually created in Altium Designer® 21 and added to our customized library to use it in the final schematic. Figure 4.28 shows the application circuit for this module according with the PIN descriptions.

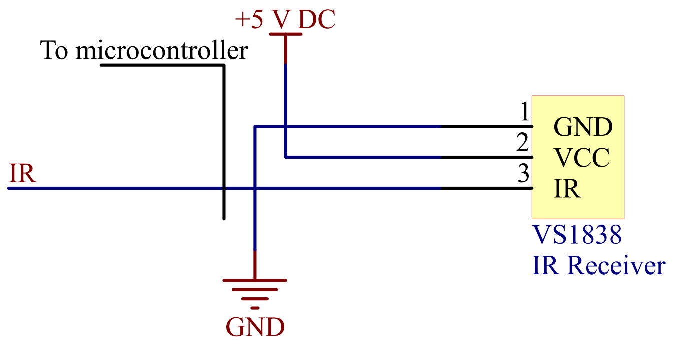
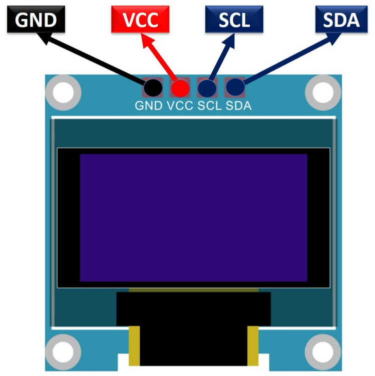


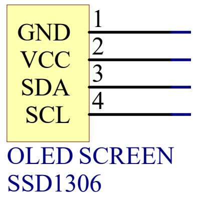
Figure 4.28 – *IR receiver module implemented circuit*

4.1.1.10 Display module

In subsection 3.2.10 we decided to use the SSD1306 OLED as the main component of the display module. Figure 4.29 (a) shows the pinout of the device which description is showed in Table 4.13.



(a) *SSD 1306 pinout [44]*



(b) *SSD 1306 schematic symbol*

Figure 4.29 – *SSD 1306 pinout and schematic symbol*

PIN(s)	SYMBOL	TYPE	FUNCTION
1	GND	POWER	Ground
2	VCC	POWER	Power supply PIN, can be connected to a 3.3 to 5 V voltage.
3	SCL	INPUT	Is a serial clock pin for I2C interface.
4	SDA	INPUT	Is a serial data pin for I2C interface.

Table 4.13 – *SSD1306 OLED display PIN description*

Figure 4.29 (b) shows the schematic symbol manually created in Altium Designer® 21 and added to our customized library to use it in the final schematic. Figure 4.30 shows the application circuit for this module according with the PIN descriptions.

- **GND (Ground PIN)** : We will connect this PIN to the common ground of the schematic.
- **VCC (Power supply PIN)**: We will power the OLED screen with +5 V DC.
- **SCL (Clock PIN)**: We will connect this PIN to the SCL corresponding PIN in the Arduino UNO.
- **SDA (Serial data PIN)**: We will connect this PIN to the corresponding SDA PIN in the Arduino UNO

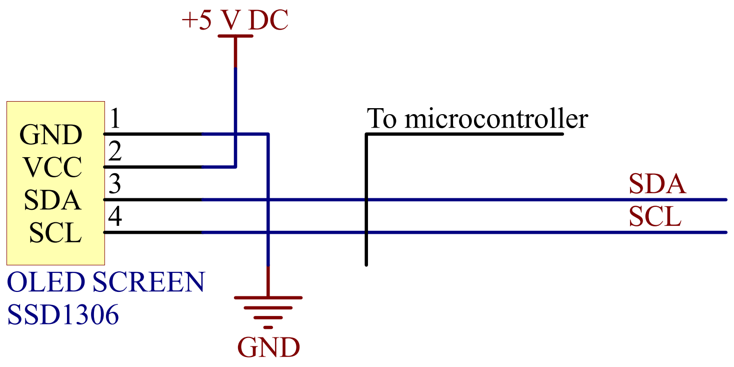


Figure 4.30 – *SSD 1306 application circuit*

4.1.1.11 Terminal connectors module

In subsection 3.2.11 we decided to use terminal blocks from phoenix contact to implement the connections for the various headlight functions to our board.

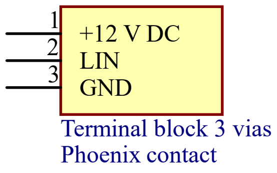


Figure 4.31 – *Terminal block Altium Designer® 21 symbol*

4.1.1.12 Final block diagram

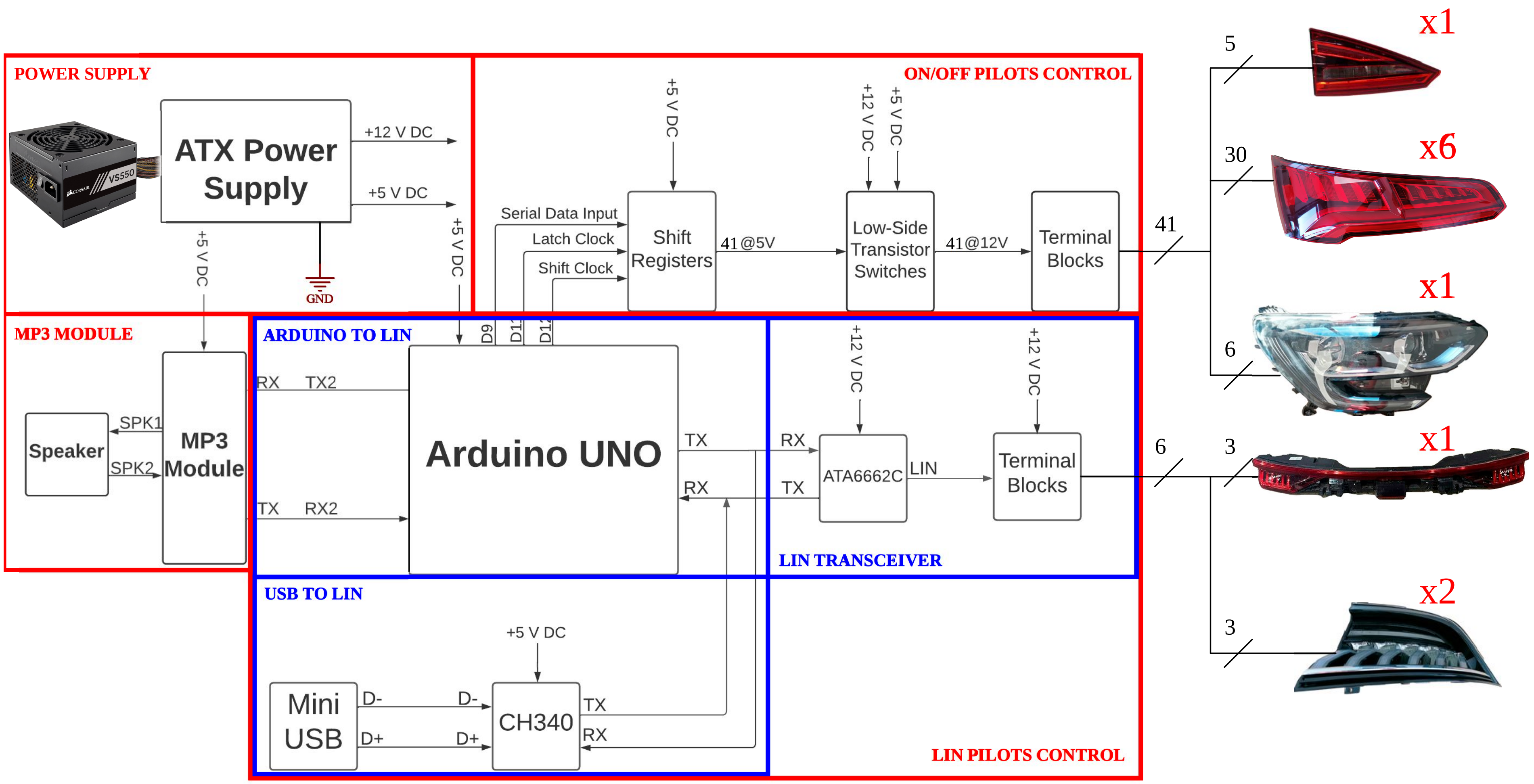


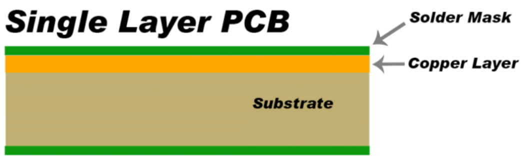
Figure 4.32 – *Final block diagram with all modules connected*

4.1.2 PCB design

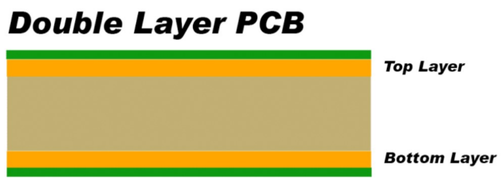
As we have already mentioned, we decided to use a PCB as the mean to implement all the electronics that we will use in this project. PCBs have a "*sandwich*" structure in which each of the conductive layers made of copper are separated from each other by insulating layers. PCBs can have a single layer, double layer, or multilayer, with the topologies that we can see in Figure 4.33. The outer copper layers of the PCB are usually covered by a layer of **solder mask**. "It is a layer made of polymer that is applied over the copper traces of the PCB to protect them from oxidation and accidental short circuits when the traces pass near large pads."

The way to connect a net of the circuit to one of the internal layers or to pass it directly from the top to the bottom layer or vice versa is carried out through what we call **vias**. A via is a vertical hole, usually small, that has copper inside (in the event that the drilling machine has the necessary technology to perform a metallic drill) and connects an outer layer with an inner layer or with the opposite outer layer.

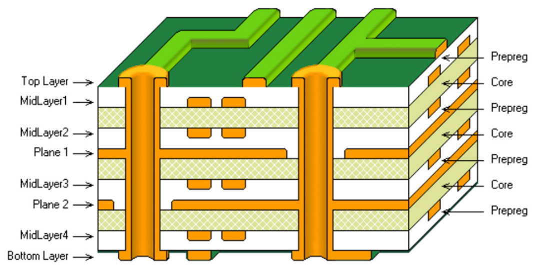
The tracks that make up the **board routing** are always located in the **outer layers** (top layer and bottom layer), while the inner layers serve as **copper planes** where we can connect the same circuit node. Ex: ground, signal, etc.



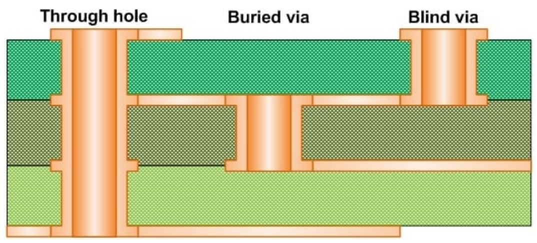
(a) *Single layer PCB topology*



(b) *Double layer PCB topology*



(c) *Multi-layer PCB stackup*



(d) *Types of vias in a PCB*

Figure 4.33 – *Kinds of PCBs and vias*

In this section, the development of the design process of a PCB will be carried out. This process consists of two parts: **schematic design and module placement** on the board. The first step has already been developed in subsection 4.1.1 using Altium Designer® 21. To design the placement and routing of the modules on the PCB we will use the same software.

The placement on the PCB of the components that make up the modules is not a simple process. We must take into account many factors such as the space occupied, the proximity of the components related to each other, the shape of the board, the number of layers, the thickness of the tracks, the design of the mechanical part, etc. The design taking into account all these factors is what will be explained in this section.

The use of Altium Designer® 21 for the design of any electronic device makes everything much easier. This is because the program automatically takes care of establishing a direct relationship between the two processes that we just mentioned, linking the "***.SCH***" files with the "***.PCB***" files, saving us the task of finding out how the PADs of each component are connected on the PCB, since the software obtains the information of the nets directly from the product schematic

In Altium Designer® 21, this relationship between the SCH and PCB files is established by relating the schematic symbols that we have shown in subsection 4.1.1 for each component with their Footprint, which is the arrangement of pads (in SMT) or through-holes (in THT) used to physically attach and electrically connect a component to a PCB. The land patterns printed on the PCB must exactly match the pinout of the component. Altium Designer® 21 establishes this relationship by looking at the pin numbering of the schematic symbol and the corresponding Footprint for that component. So we must be very careful when designing the footprints, because if it is not correctly related to the SCH symbol, the circuit will not work.

As a criterion for designing the Footprints of our components, we can highlight making them as similar as possible in size to the real component. Also, at the bottom of most component Datasheets, there are some recommended dimensions for it's Footprint.

4.1.2.1 Steps for a good PCB design

If we want the design of our PCB to be efficient and of quality, we must make a selection considering the following points:

- **PCB production technology**: The first and most important decision that we must make as a basis for developing a good design is the technology that will be used to manufacture the PCB. In our case, we have the possibility of manufacturing the plate in GranaSAT's own facilities or ordering its manufacture from China so that they can send it to us. In this case, we have decided to carry out both options to be able to later be in a position to make a comparison. In chapter 5 the pros and cons of each PCB fabrication technology are shown. In the case of the order to China, the `jlcpcb.com` [25] website has been chosen.

- **PCB features**: Once we have decided on the manufacturing technology offered by the PCB manufacturer, we must define the capabilities that our PCB requires to ideally adapt to our design needs.

Layers	2
PCB Qty.	5 (mín.)
Material	FR-4 Standard Tg 130-140C
Dielectric constant	4.5 (double-side PCB)
Dimensions	126.75 x 92.71 (mm)
Board thickness	1.6 mm
Finished outer layer cooper	1 oz
PCB color	Green
Silkscreen color	White

Table 4.14 – *PCB features chosen in `jlcpcb. com` [25] manufacturer*

We have decided to use a double layer PCB since, taking a look at the complexity of the schematics available in section B.1. We believe that a single layer topology would be too detrimental to the design in terms of space utilization. On the other hand, the use of a multilayer topology could be excessive, since the design does not have so many important nets to assign them a specific plane.

The number of PCBs to be manufactured will be 5, since it is the minimum that the manufacturer `jlcpcb.com` allows to order, since it is a wholesale seller.

- **PCB design rules**: Whatever the environment in which we design a PCB, there are certain limitations imposed by **design rules**. Compliance with these rules is mandatory for the PCB to function properly and smoothly. In our case, the design rules imposed by the machine available in the GranaSAT laboratory are, logically, much more limited than those offered by the Chinese manufacturer at `jlcpcb.com` [25].

  Some of the parameters that define the design rules are, for instance: the space needed between each element, the maximum an minimum hole size, the minimum and maximum track size, etc. Some of the design rules imposed by `jlcpcb.com` are shown in Table 4.15.

Features	**Capability**
Max. dimensions	400 x 500 mm
Board thickness	0.4/0.6/0.8/1.0/1.2/1.6/2.0 mm
Layer count	1, 2, 4, 6
Drill Hole Size	0.20 mm - 6.30 mm
Pad Size	1 mm mín.
Via to Via clearance(Same nets)	0.254 mm
Via to Track clearance	0.254 mm
Mín. trace width	5 mil

Table 4.15 – *Some PCB design rules imposed by `jlcpcb.com` [25] manufacturer*

- **Mounting technology:** Once we know the design rules that we must adhere to and the characteristics that we have finally chosen, we must choose the mounting technology that we will use for the PCB components. The vast majority of the components that we have needed for our project are available in SMD technology, which presents clear advantages over THT in terms of weight and size. That is why we have decided to choose, whenever possible, the SMD version of the component in question. Some of the components used only have a THT version, such as the VS1838 IR receiver, the SSD 1306 OLED screen, the Arduino UNO and the MP3 player.

- **Package types:** Just as there are different types of mounting technology for each component, there are also usually different sizes available. In the case of our choice (prioritizing the use of SMD components), we must choose which package size best suits our needs and limitations. For example, in the case of passive components such as capacitors and resistors, there are packages of 1005 [2], 0201, 0402, 0603, 0805, 1008, etc.

  To make a decision regarding this, the main factor that we must take into account is the way in which the components are going to be soldered to the board. In our case, even if we order the PCB from China, it will come, for obvious reasons, with the components unsoldered, a task that we will have to carry out in the GranaSAT laboratory. It is for this reason that we are not interested in using a package that is too small that would make the soldering task too difficult. Likewise, we are not interested in too large a package that increases the space needed on the PCB and, therefore, the cost of manufacturing it and its weight. We must point out that, along with the PCB, `jlcpcb.com` also offers us a Stencil to facilitate the soldering of the components through the use of solder paste, so we will solder them in this way. Finally we have decided to make general use of **0805 packages**, since they fulfill the commitment of not being too big to affect the size of the board nor too small to make placing difficult.

[2]Package type numbering refers to their size in imperial units, which in this case is thousandths of an inch (mil). The first two digits indicate the length and the last two the width. For example, a package that is 0805 means that the component dimensions are 8 x 5 mil.

4.1.2.2 PCB's modules placing

The process of selecting the distribution of the different modules described throughout this work may seem simple and unimportant. However, the distribution of the components is so important that it will be a determining factor in the efficiency and quality of our product.

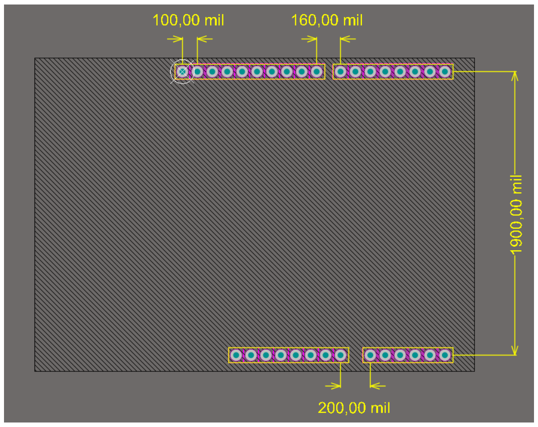
Facilitating subsequent component routing is the most important criterion to consider when thinking about how to distribute modules across the PCB. Also, in order to make our product as comfortable and accessible as possible, there are some components that need to be located in specific areas of the PCB like connectors, which must be at the edge of it.

Next, we will make a brief qualitative description of the criteria that we have followed to choose in which area of the PCB we have decided to place the different modules:

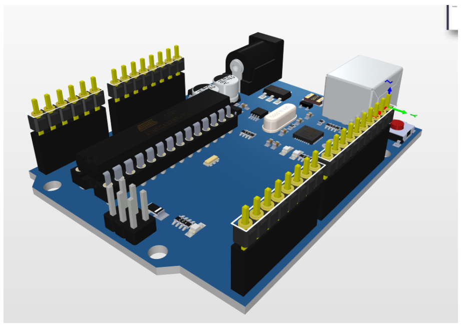
- **LIN transceiver module**: The main component of this module is the ATA6662C and it is used to transform the serial communication coming from the USB to serial module or the Arduino into LIN communication. The logical place where it would make sense to place this module is near the two previously mentioned modules.
- **USB port module**: This module consists only of the mini USB type B connector. We will place it somewhere on the edge of the board and near the USB to serial module.
- **USB to serial module**: This module is responsible for converting the communication from the differential pair of the USB connector module to serial communication to take it to the LIN transceiver module, so the best option is to place it near both.
- **Shift register module**: This is the module in charge of increasing the number of digital outputs available in the Arduino UNO microcontroller to control the gates of the N-channel MOSFET transistors present in the power amplifier module, which is why it would be interesting to place the 4 registers online in some area near both modules.
- **Audio player module**: This module consists only of the DF-Player mini. When choosing the best place on the board to place it, we must bear in mind that this component incorporates the possibility of using a microSD card, so it must necessarily be placed on the edge of it. Also, if possible, we could place it near the selected Arduino UNO pins as VSP dedicated to this module.
- **Power amplifier module**: The topology of this module will be repeated as many times as there are functions in total coming from the headlights (41). It is in charge of raising the power available in the Arduino UNO microcontroller pins to be able to turn on the different functions of the headlights. It makes sense to place each chain of NMOS-PMOS transistors in line with each output pin of the terminal blocks. In this way the subsequent routing will be much simpler.
- **Remote control module**: This module consists only of the IR receiver whose objective is to capture the IR signals generated by the remote control, decode them and send them to any digital PIN of the Arduino UNO. It makes sense to place this module somewhere on the edge of the board so as not to hinder the capture of the IR signal (remember that direct vision between the emitter and receiver is needed) and also, if possible, it would be good to place it near the Arduino UNO microcontroller.
- **Display module**: This module, logically, being a screen in which we will see information, must be located in a place on the PCB that is not physically covered by any component and that is visible to the naked eye. Also, as far as possible, it would be interesting to place it somewhere near the and pins of the Arduino UNO microcontroller.
- **Terminal connectors module**: The terminal blocks to which we are going to connect each wire corresponding to the functions of the headlights must be located on the edge of the board so that they can be easily accessible at all times.

4.1.2.3 Components footprints and 3D-models

This section shows all the Footprints with its dimensions and 3D models corresponding to each of the components placed in the PCB. The numbering of the leads must match with the pinout of the SCH symbol of each component presented in subsection 4.1.1.

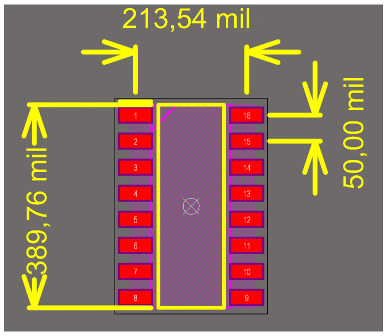


(a) *Footprint*

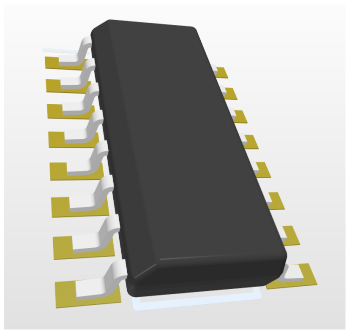


(b) *3D model*

Figure 4.34 – *Arduino UNO PCB model*

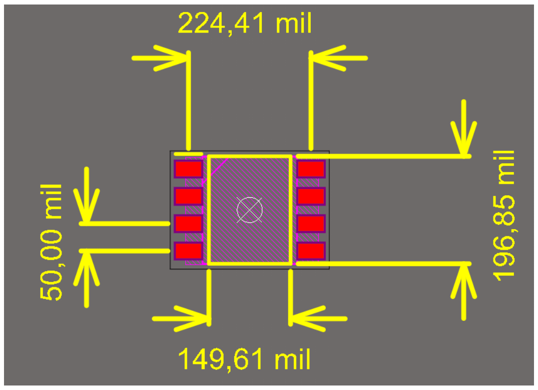


(a) *Footprint*

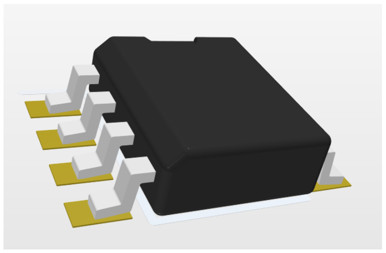


(b) *3D model*

Figure 4.35 – *SOP-16 package PCB model*

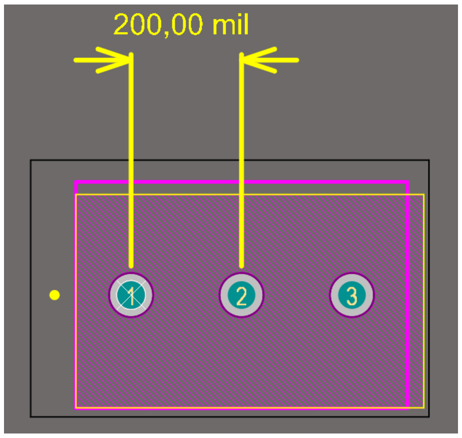


(a) *Footprint*

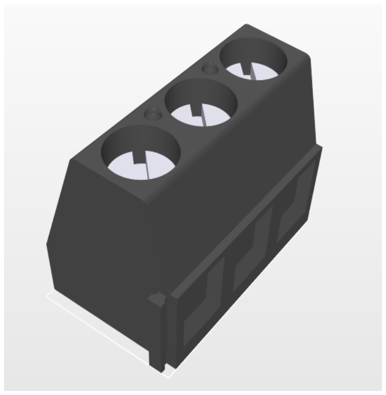


(b) *3D model*

Figure 4.36 – *SOP-8 package PCB model*

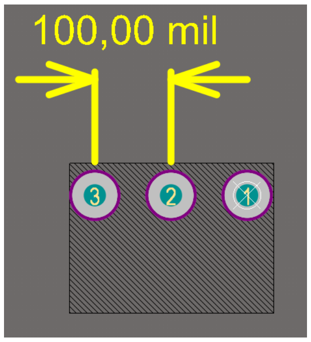


(a) *Footprint*

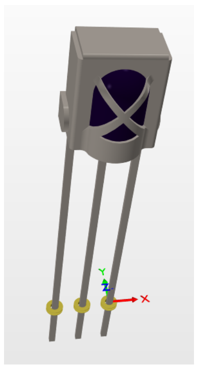


(b) *3D model*

Figure 4.37 – *Terminal block PCB model*

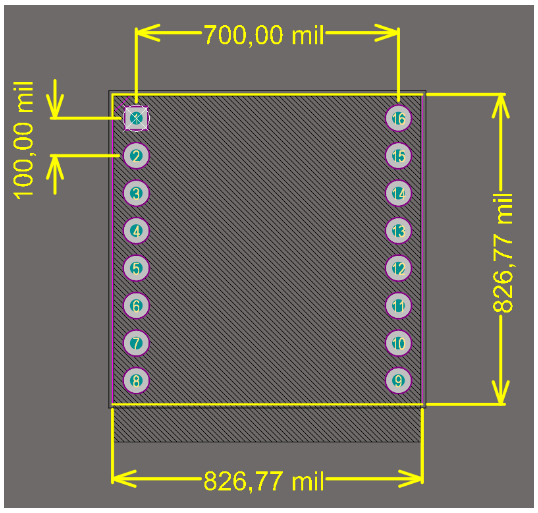


(a) *Footprint*

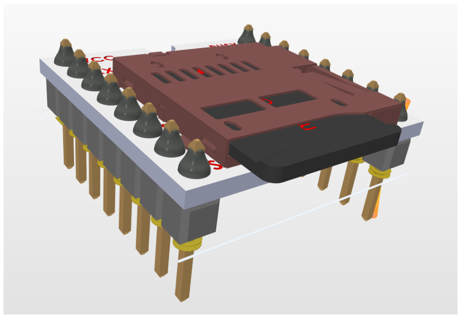


(b) *3D model*

Figure 4.38 – *VS1838 package PCB model*

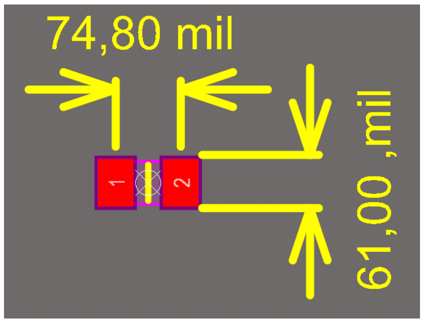


(a) *Footprint*

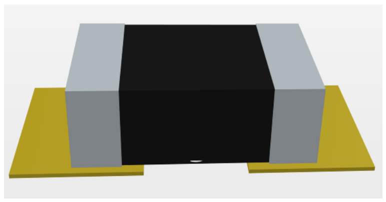


(b) *3D model*

Figure 4.39 – *DF-Player mini PCB model*

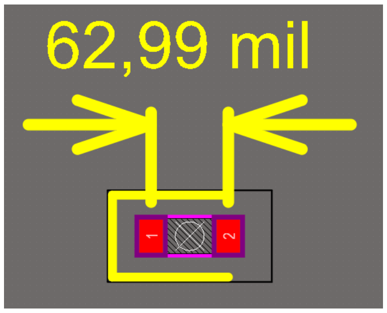


(a) *Footprint*

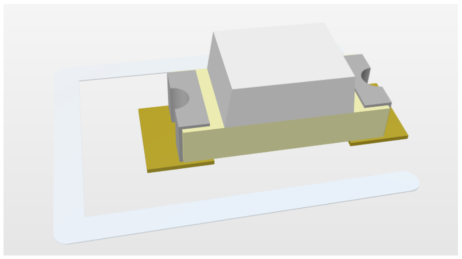


(b) *3D model*

Figure 4.40 – *0805 resistor PCB model*

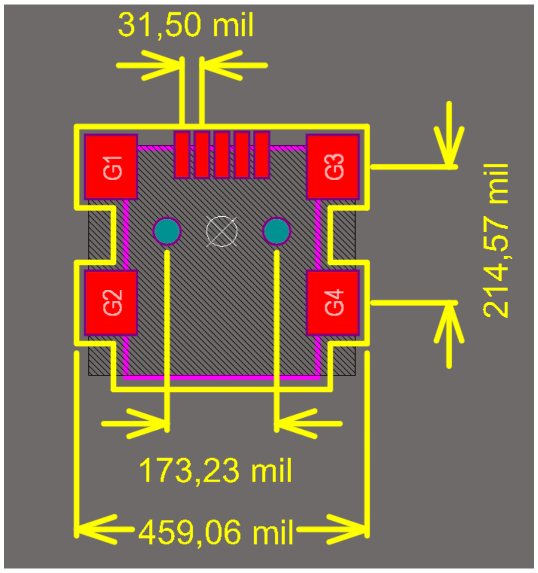


(a) *Footprint*

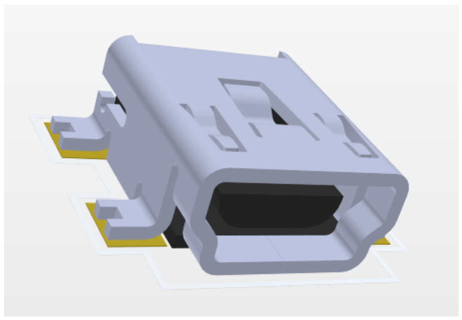


(b) *3D model*

Figure 4.41 – *0603 LED PCB model*

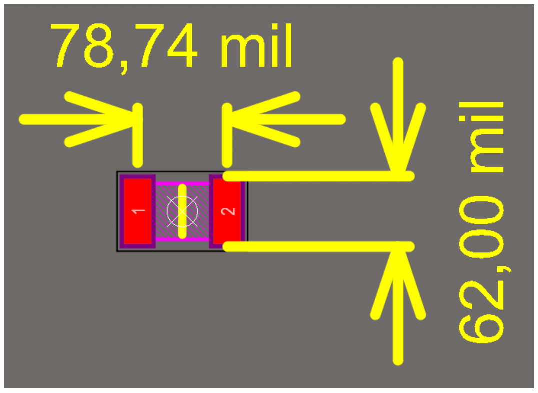


(a) *Footprint*

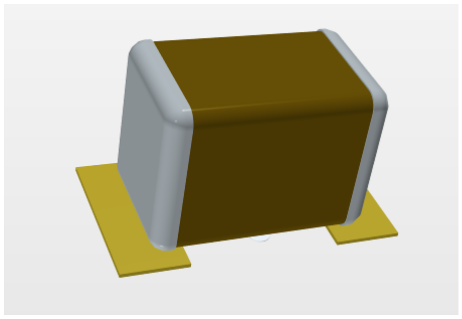


(b) *3D model*

Figure 4.42 – *USB type B PCB model*



(a) *Footprint*



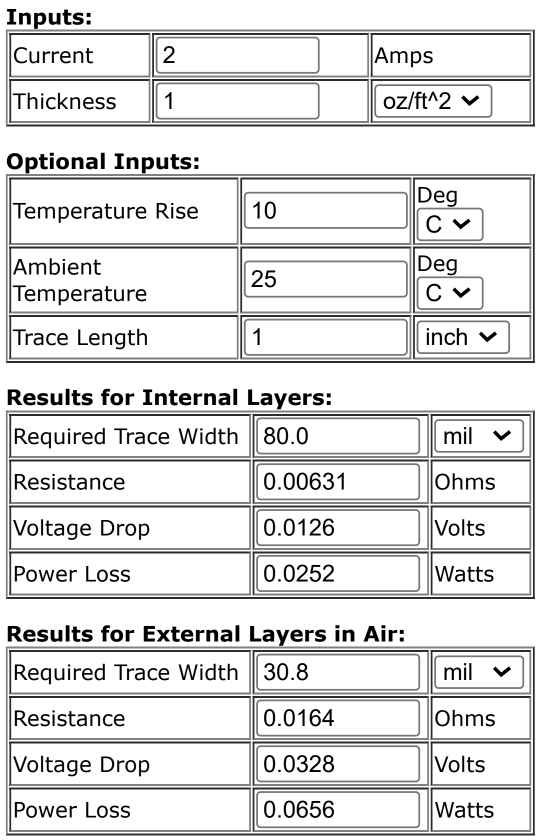
(b) *3D model*

Figure 4.43 – *0805 capacitor PCB model*

4.1.2.4 PCB routing

Before placing the components on the board, we must analyze their routing needs, since not all components need the same track width and, furthermore, there are some connections that need to be made in a specific way for their correct operation:

- The USB tracks are part of a "**differential pair**", so they must be exactly the same length, since the information carried by the USB resides in the potential difference between the two tracks.
- The current that circulates through the tracks that are connected to the terminal blocks is the current that the function of the headlight demands in the context to which they are connected. As we can see in section E.1, some of these functions require a very high current and, therefore, their corresponding track would have to be able to withstand it without overheating too much. Figure 4.44 shows the use of a calculator present on `www.4pcb.com` to calculate the recommended minimum width of the tracks according to the **current** that will circulate through them and their **thickness**.


Inputs:

Current	2	Amps
Thickness	1	oz/ft^2

Optional Inputs:

Temperature Rise	10	Deg C
Ambient Temperature	25	Deg C
Trace Length	1	inch

Results for Internal Layers:

Required Trace Width	80.0	mil
Resistance	0.00631	Ohms
Voltage Drop	0.0126	Volts
Power Loss	0.0252	Watts

Results for External Layers in Air:

Required Trace Width	30.8	mil
Resistance	0.0164	Ohms
Voltage Drop	0.0328	Volts
Power Loss	0.0656	Watts

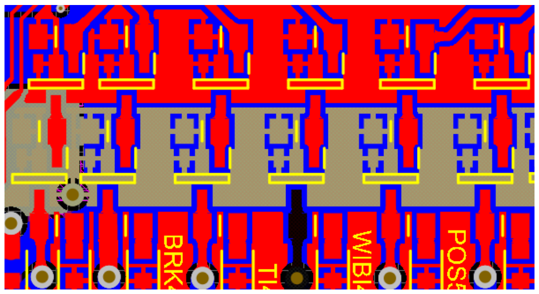
Figure 4.44 – *PCB tracks width calculator [48]*

Doing some quick calculations, some track widths corresponding to different headlight functions are shown in the Table 4.16. As we can see, some of the functions require a much larger track width than the others. This has been taken into account when performing the routing of the PCB. In general, the power tracks have been made thicker than the signal tracks, always staying within the limits of the design rules.

Headlight	Function	Current (A)	Trace width (mil)
Fender Audi Q8 2018	Brake	0,4	3,34
	TI	0,25	1,75
	Position	0,26	1,84
Headlight Renalut Megane 2016	DRL	1,36	18,1
	HB	3,9	77,3
	LB	3,9	77,3
Trunk Audi Q8 2018	Brake	0,34	2,67
	TI	0,3	2,25
	Position	0,54	5,06
	Back	0,79	8,54

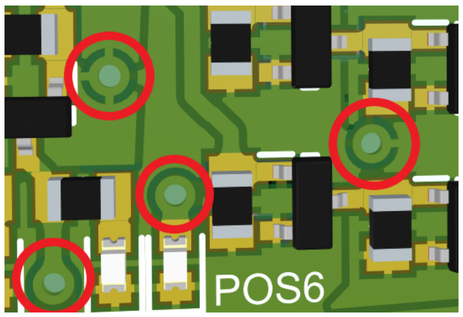
Table 4.16 – *Calculated tracks width corresponding to functions current consumption*

In addition, some routing techniques have been used that make our design improve in terms of quality, durability and efficiency, these are:

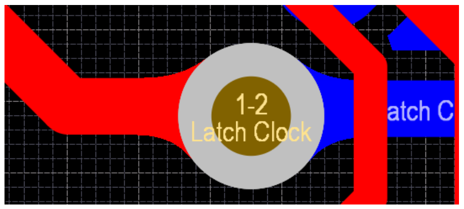


(a) *Polygons*

(b) *Via stitching*



(c) *Via shielding*



(d) *Teardrops*

Figure 4.45 – *PCB routing techniques used*

- **Use of polygons:** Altium Designer® 21 gives us the possibility to create polygons with the shape we want and assign them a net. In this way, the entire area that the polygon occupies will be a copper sheet connected to the net that we have assigned. This is infinitely useful for routing power nets like +12V, +5V and GND, which are all over the PCB and would be tedious to manually route them with tracks. Figure 4.45 (a) shows an example of polygons used for the net of GND (the red one) and +12 V (the yellow one). The space applied between several polygons or between polygons and routes or vias is also defined in the design rules.
- **Via stitching:** Once we have finished our routing and polygon design on both the TOP and BOTTOM layers, Altium Designer® 21 gives us the option to apply via stitching. This technique consists of adding a multitude of vias that connect the different polygons present in the top layer with those that have the same net assigned in the bottom layer. In this way, the power nets will be much better connected, since they will have a greater area of contact with each other, which is very good for distributing the temperature and avoiding overheating. An example of the use of this technique in our PCB can be seen in Figure 4.45 (b).
- **Via shielding:** This process consists of applying solder mask on top of the copper around the vias on the PCB. This is a good thing to do, as copper exposed to the outside of PCBs tends to oxidize and deteriorate over time. As our PCB does not have a box and is designed to be screwed to the wooden board where the headlights are, it is convenient to carry out this procedure, which we can see in Figure 4.45 (c).
- **Teardrops** [3]**:** A teardrop is extra copper, straight or rounded, at the junction of a pad or via and a trace, or when a trace transitions between thick and thin. For example, if part of your trace width is changing from 30 mils to 10 mils, then the teardrop is added at the transition point to reduce any stress or hairline cracks.

4.1.2.5 PCB final design

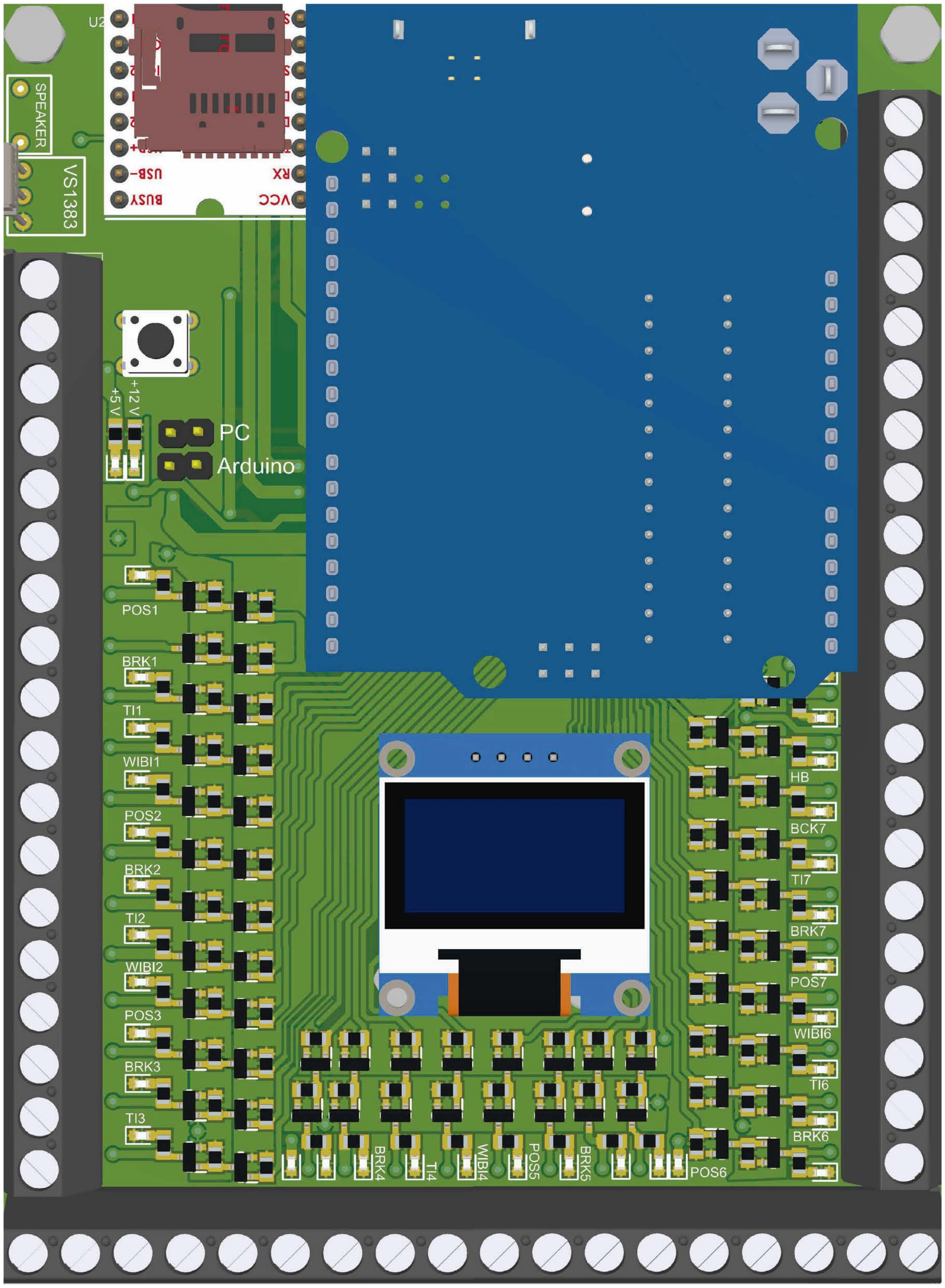


Figure 4.46 – *PCB final design 3D top view*

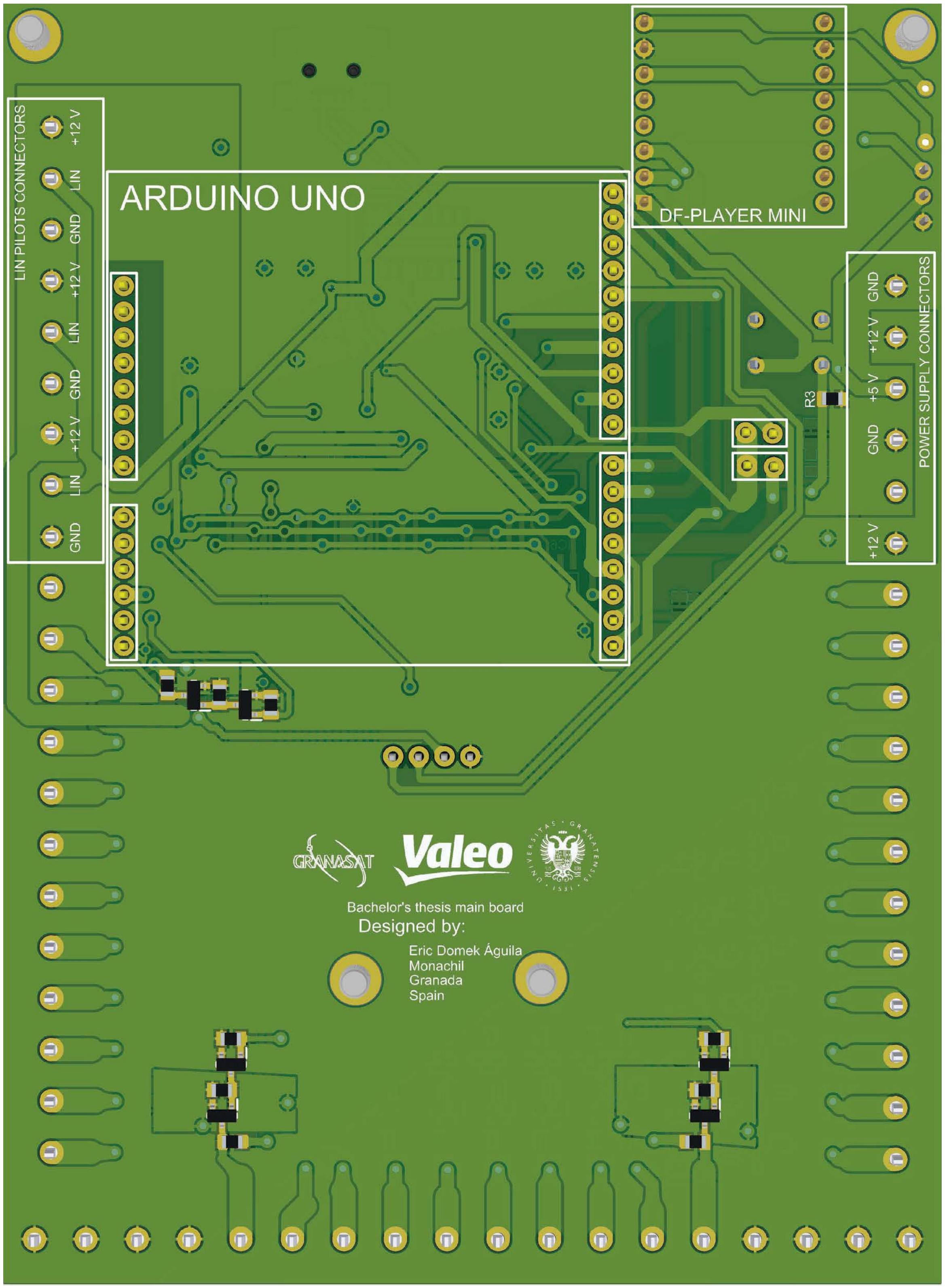


Figure 4.47 – *PCB final design 3D bottom view*

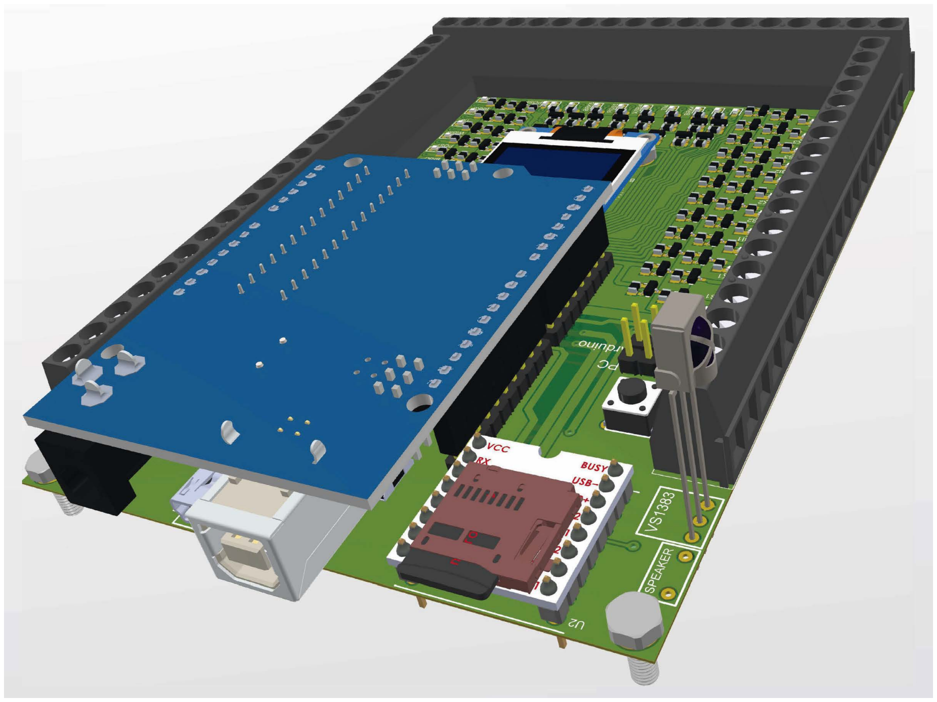


Figure 4.48 – *PCB final design 3D side top view*

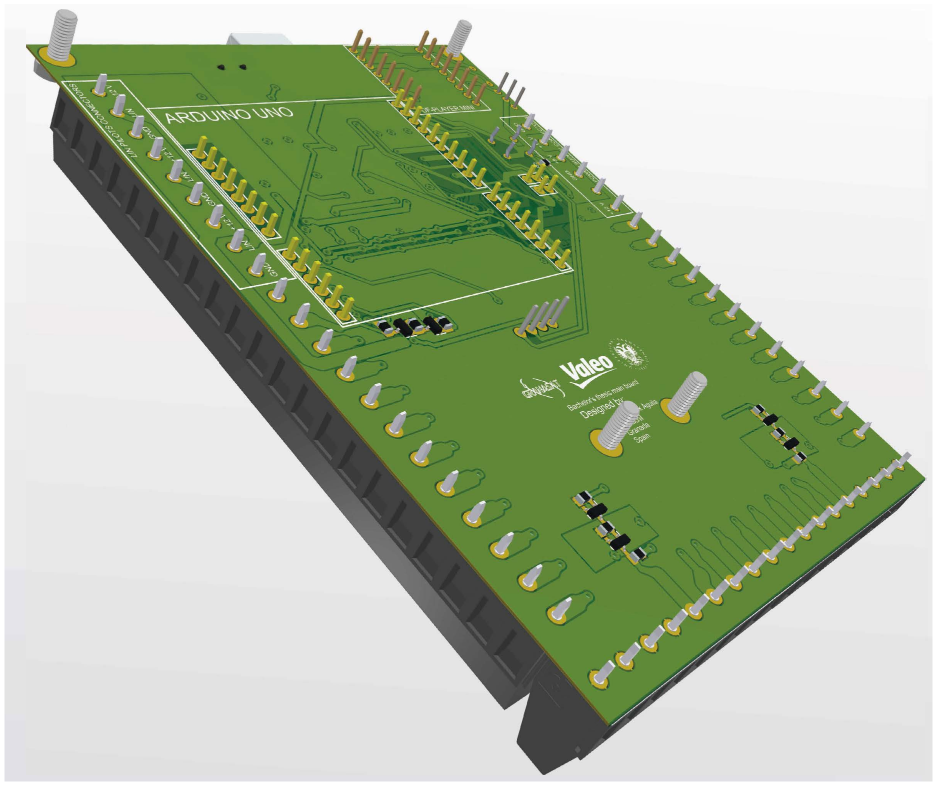


Figure 4.49 – *PCB final design 3D side bottom view*

4.2 Mechanical design

4.2.1 Demonstrator plank design

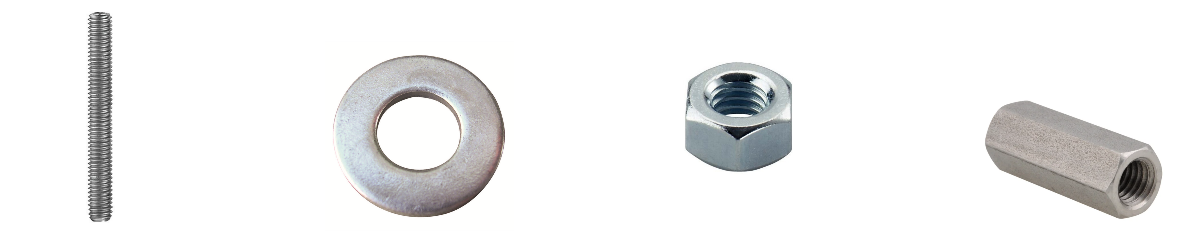
In section 3.3 we analyzed the anchoring method available to each type of headlight in particular. We measured the **dimensions** of each headlight, in addition to the **pitch and metric** of the screws. In this section we will detail the way in which the headlights will be anchored to the wood, detailing the necessary materials in each case and showing the final distribution of the wooden panel.

In order to get an idea of what our designs would look like before carrying them out in reality, we have decided to use the software SolidWorks®, which will allow us to create 3-D pieces corresponding to the different real materials that we will use, as well as use them to create assemblies that will offer us a fairly approximate version of how our design will finally look.

4.2.1.1 Trunk (long), Trunk Lamp and Trunk LHD anchoring

Depending on the specific anchoring method of each headlight, we will need different materials. As we saw in subsubsection 3.3.1.1, the 6 ***"Trunk Lamps"*** have 3 **M5 screws** with a **pitch of 0.8 mm**; the ***"Trunk LHD"*** has 3 **M6 screws** with a **pitch of 1 mm** and the ***"Trunk (long)"*** has got 8 **M6 screws** with a **pitch of 1 mm**. To anchor these headlights we will use the materials from the following list that we can see in Figure 4.50:

- 36x **M5 nuts**
- 36x **M5 washers**
- 18x **M5 threaded unions**
- 18x pieces of **M5 threaded rod**
- 11x **M6 threaded unions**
- 11x pieces of **M6 threaded rod**
- 22x **M6 washers**
- 22x **M6 nuts**

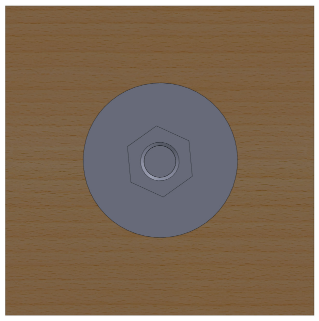


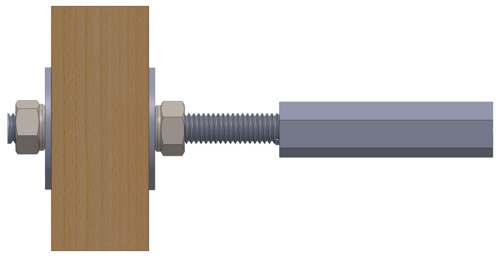
(a) *Threaded rod* **(b)** *Washer* **(c)** *Nut* **(d)** *Threaded union*

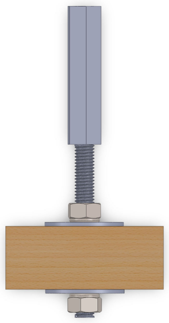
Figure 4.50 – *Necessary elements for anchoring **"Trunk Lamps"**, **"Trunk (long)"** and **"Trunk LHD"** headlights*

Figure 4.51 shows the method that has been used to anchor this type of headlights to the wooden panel.

It is obvious that, although the anchoring method is completely analogous, for each case the materials of the corresponding measurements have been used.


(a) *Front view*


(b) *Side view*


(c) *Top view*

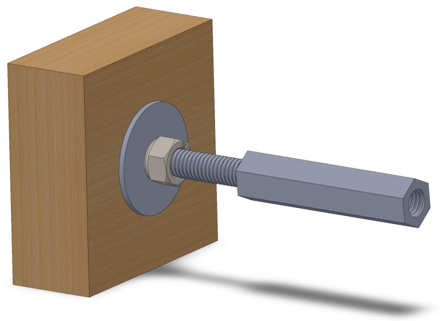

(d) *Angled view*

Figure 4.51 – *Anchoring method for* ***"Trunk Lamps"****,* ***"Trunk (long)"*** *and* ***"Trunk LHD"*** *headlights recreated in SolidWorks®*

It is important to note that we have not been able to find M5 union sleeves in our trusted hardware store. That is why we have had to create our own in the laboratory from a material called "white plastic".

4.2.1.2 Fender and forward lamp anchoring

As these headlights do not have male screws, the anchoring to the wooden plank is done differently. As we can see in Figure 4.52 (a) and Figure 4.53 (b), these headlights have got some holes through which we can introduce, for instance, a threaded rod. In order not to have to buy more materials, we have verified that the M6 threaded rods fit quite well in these holes, so we will also use them to anchor the fender lights of the Audi Q8.

For the headlight of Renault Megane 2016 we decided to use some screw-in hooks, because some of the holes in this headlight are oriented horizontally, which makes it impossible to use threaded rods as in the previous cases. In addition, this headlight has a very notable difference in weight compared to the rest, and this new type of anchorage is more resistant.

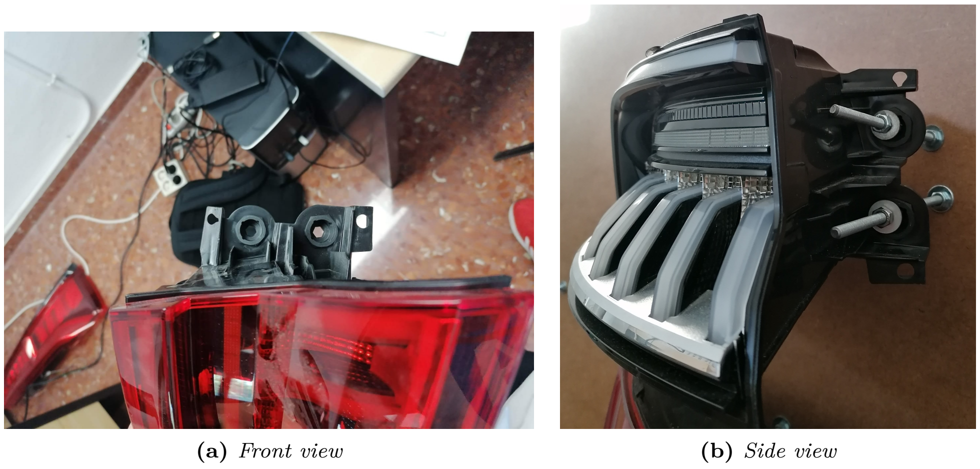

(a) *Front view* **(b)** *Side view*

Figure 4.52 – *Anchoring method of Fender light of Audi Q8 2018*

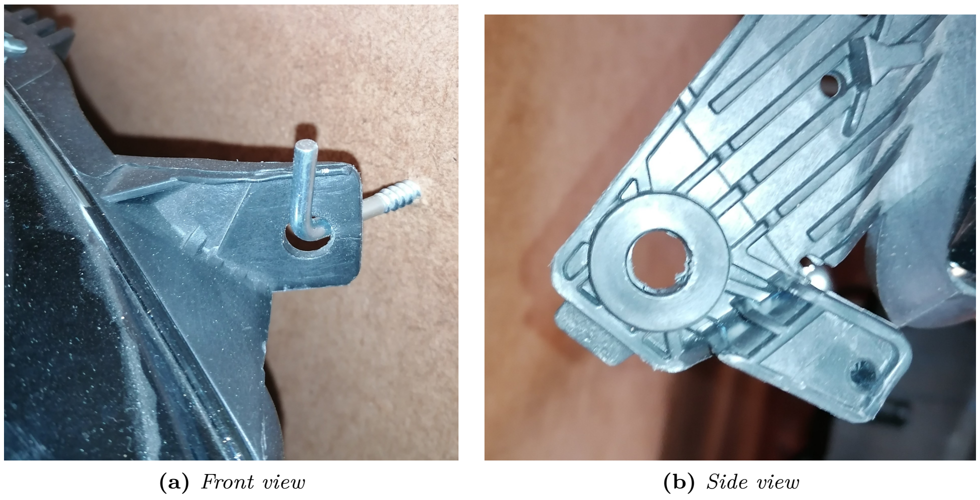

(a) *Front view* **(b)** *Side view*

Figure 4.53 – *Anchoring method of "**Renault Megane 2016**" headlight and "**Audi Q8 2018**" Fender light*

4.2.1.3 Final distribution

Once we have the measurements of the dimensions of all the headlights and we have designed the method of anchoring each one, we can proceed to design the final distribution of the headlights on the wooden plank. It is important to note that it will always be preferable to choose the smallest possible size for the wooden plank in order to reduce its weight and be able to transport the product as comfortably as possible.

To get an idea of how the final distribution would look like, once again we have used the software SolidWorks®. Specifically, for each particular headlight we have created a "box" that has the same measurements as the headlight itself. In this way it will be easier for us to get an idea of the three-dimensional space that each headlight occupies and we will be able to choose a suitable size for the wooden plank that allows us to distribute the headlights comfortably.

On the next page we can see the Figure 4.55 showing the distribution of headlights recreated in SolidWorks® from various points of view.

As we can see, the number of headlights we have and the measurements of each of them have conditioned the size of the wooden plank. The ideal thing would be to have a custom size that perfectly fits the idealized distribution. Unfortunately, there are standard sizes of wood planks on the market, which means that most businesses stick to them. Table 4.17 shows some of the standard MDF planks dimensions.

WIDTH (cm)	HEIGHT (cm)	THICKNESS (cm)
366	207	2 to 60
305	122	2 to 60
244	**122**	**2 to 60**
285	210	2 to 60
280	207	2 to 60

Table 4.17 – *Standard MDF wooden plank dimensions [7]*

Luckily, a **244 x 122 cm board** is pretty close to the ideal size we need for the headlight layout we've designed, so we'll stick with that one.

In terms of **thickness**, the board should be thick enough for the anchors to be firm; but not too thick, as it would add a lot of weight to the product and make it more difficult to transport. Based on these observations, we have decided that **a thickness of 1.9 cm** is a good consensus.

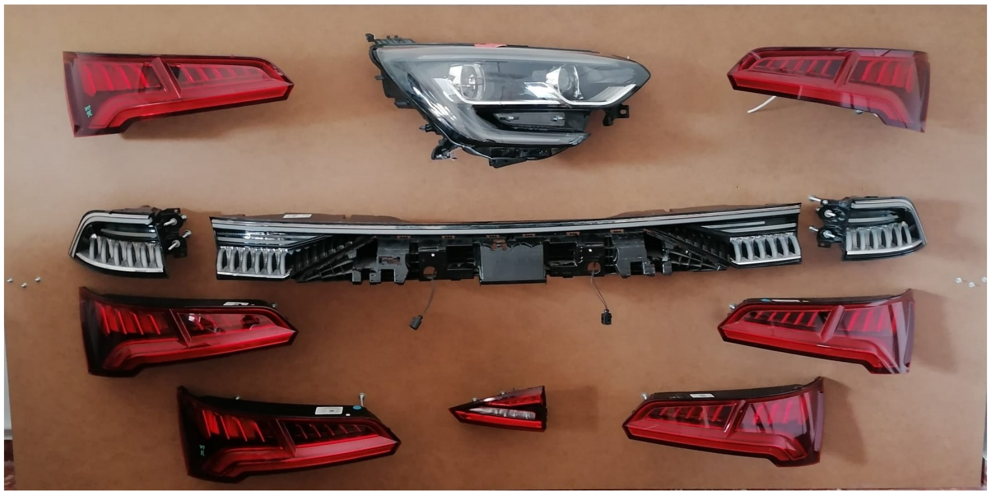
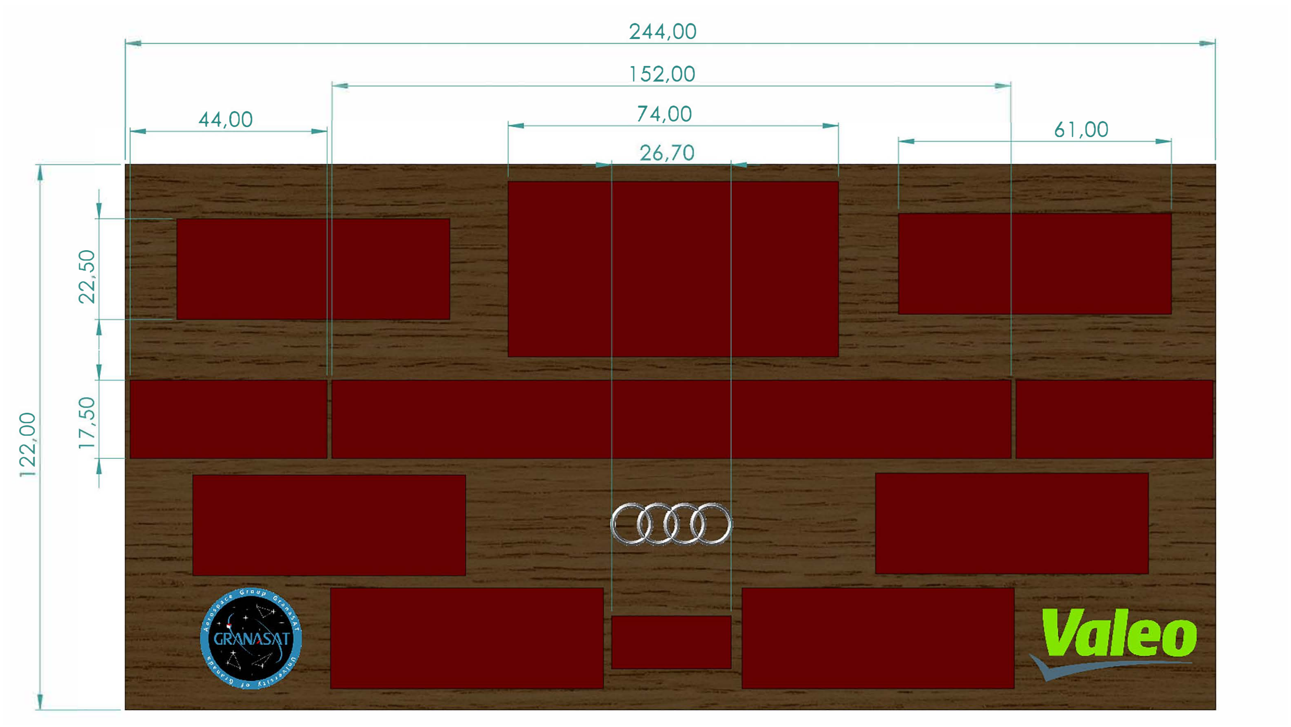
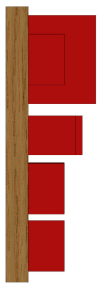


Figure 4.54 – *Wooden plank final result (front view)*

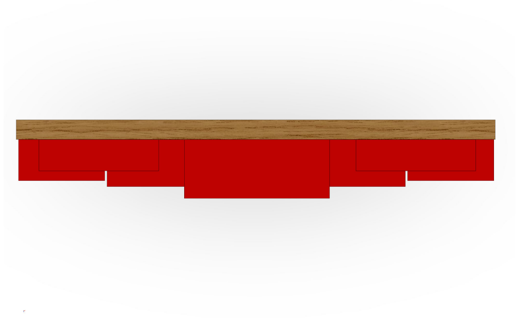
Figure 4.54 shows a front view of the final result of the wooden plank once each and every one of the headlights we had had been anchored. It is noticeable that the final appearance is very similar to the one planned in SolidWorks®. So we conclude that the use of SolidWorks® software has been key to planning the construction of the mechanical part of the project.



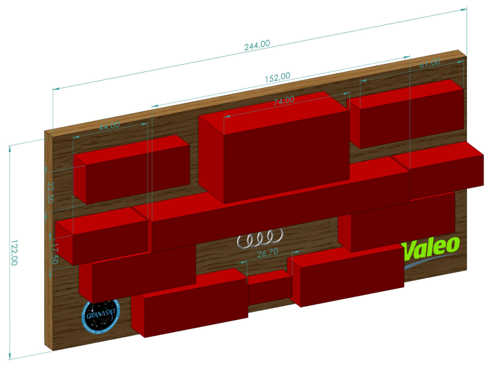
(a) *Front view*



(b) *Side view*



(c) *Top view*



(d) *Angled view*

Figure 4.55 – *Various views of headlights distribution at wooden plank recreated in SolidWorks®*

4.3 Software Design

When we design the software of a new product, we must first make sure that each module of which it is composed performs its function correctly separately. Only then can we write the full Firmware of the product. That is why in this section we will show the pieces of code that each module needs to work correctly. Examples of operation of each module will also be shown separately.

4.3.1 SSD 1306 OLED screen code

To meet the requirement "***Offering a user-friendly interface through which you can carry out control of all functions.***" specified in section 2.1, we decided to include the SSD 1306 OLED screen in our project. In this section we will design the code fragment in C language that the Arduino must execute to implement a menu on this screen.

First of all, we must load the libraries corresponding to the Adafruit SSD 1306 to be able to use the necessary commands, these libraries are "***Adafruit_GFX.h***" and "***Adafruit_SSD1306.h***". In addition, we must include the "***Wire.h***" library, which will allow the Arduino UNO to communicate with I2C/TWI devices.

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
```

Listado 4.1 – *Inclusion of necessary libraries for the operation of the OLED SSD 1306 screen*

Next, it is necessary to define some necessary variables so that the "***Adafruit_SSD1306.h***" library knows some data about the screen that we are using, such as the **width**, the **height**, the Arduino UNO PIN that we will use as **RESET** and the **address** in which it is located within the I2C bus. Once defined, we must execute a function to load them into the library

```
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define OLED_RESET     4 // Reset pin # (or -1 if sharing Arduino
 reset pin)
#define SCREEN_ADDRESS 0x3C ///< See datasheet for Address; 0x3D for
  128x64, 0x3C for 128x32

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
 OLED_RESET); // We load the data defined in the Adafruit function
```

Listado 4.2 – *Definition of the necessary variables for the correct control of the OLED screen*

We will now define and initialize two variables needed to implement navigation within the menu. In the next sections we will show the role played by these variables.

```
int selected = 0; // This variable will serve to define the position
  in which we are within the menu
```

```
int entered = -1; // This variable will serve to define the option
  chosen from the menu
```

Listado 4.3 – *Selected and entered variables definition and initialization*

We will now move on to defining the code's setup function. The setup is the first function to be executed within an Arduino UNO program. It is basically where the functions to be carried out by the microcontroller are "set". This is where we set some criteria that require a single run.

Basically what this piece of code does is:

- Defining digital pins 2 through 5 as input pins. In our final project we will use the IR remote control to control the menu, but for this example we have decided to implement its control through 4 buttons to make it more clear and don't mix code from multiple modules.
- Initialize serial communication at 9600 baud.
- Check that the Arduino UNO has managed to find the screen on the I2C bus successfully.
- Draw a single yellow pixel on the screen for 2 seconds to indicate that the screen was found successfully

```
void setup() {
  pinMode(2, INPUT_PULLUP);
  pinMode(3, INPUT_PULLUP);
  pinMode(4, INPUT_PULLUP);
  pinMode(5, INPUT_PULLUP);

  Serial.begin(9600);

  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V
 internally

  if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;); // Don't proceed, loop forever
  }
  display.clearDisplay();
  // Draw a single pixel in yellow
  display.drawPixel(10, 10, SSD1306_WHITE);
  display.display();
  delay(2000); // Pause for 2 seconds
}
```

Listado 4.4 – *Setup function*

Below is the loop function from the code. This function does precisely what its name suggests, and loops consecutively, allowing the program to change and respond. In this case, all we want this function to do is periodically call the function created to display the menu on the screen.

```
void loop() {
  displaymenu(); // Initialize and display the functional menu on
 the OLED screen
}
```

Listado 4.5 – *Loop function*

Finally, the most important part of the code to make the screen work is the function "***displaymenu***", which implements the working menu.

First of all, within this function we must define the variables corresponding to the functions that are going to implement each of the 4 buttons that have been used for this example. What will be stored in each of these variables when we press its corresponding button will be a "***LOW***" value. While by default they have a "***HIGH***" value, since the push button has been implemented with a pull-down resistor.

A set of conditions are defined that implement the following behavior:

- If the ***UP*** button is pressed, the "***selected***" variable is incremented by 1.
- If the ***DOWN*** button is pressed, the "***selected***" variable is decreased by 1.
- If the ***ENTER*** button is pressed, the "***entered***" variable takes the value of "***selected***".
- If the ***BACK*** button is pressed, the "***entered***" variable becomes -1.

```
void displaymenu(void) {

  int down = digitalRead(3);     // Move down one position in the
 menu
  int up = digitalRead(2);       // Move up one position in the menu
  int enter = digitalRead(4);    // Enter the selected option
  int back = digitalRead(5);     // Goes back to the main menu

  if (up == LOW && down == LOW) {
  };
  if (up == LOW) {
    selected = selected + 1;
    delay(200);
  };
  if (down == LOW) {
    selected = selected - 1;
    delay(200);
  };
  if (enter == LOW) {
    entered = selected;
    delay(200);
  };
  if (back == LOW) {
    entered = -1;
  };
```

Listado 4.6 – *Polling based measurements script*

The following piece of code defines the options that the menu will display on each screen. This is done by defining a pointer string for each of the displayed screens.

```
const char *options[4] = {
        " Preset shows",
        " LIN pilots",
        " ON/OFF pilots",
        " Credits ",
};

const char *options_shows[4] = {
        " Show 1",
        " Show 2",
        " Show 3",
        " Show 4",
};

const char *options_LIN[5] = {
        " Position",
        " Animated Tail",
        " Dynamic Tail",
        " Reverse",
        " Stop",
};

const char *options_ON_OFF[4] = {
        " Fender lights",
        " Trunk Lamps",
        " Forward lamp",
        " Trunk LHD",
};
```

Listado 4.7 – *Definition of the character strings that will appear in each menu screen*

The following defines what happens when each option present in the main menu is selected. In general, what happens is:

- The screen is cleared and the title of the project "*PILOT CONTROL DEVICE*" is written as header.
- The name corresponding to the menu option to which we have accessed is written.
- Through a loop that runs through the corresponding character string, each of the options accessible from that menu is written in the form of a list.

```
    // If the selected option is -1, it returns to the main menu
```

```
  if (entered == -1) {
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(0, 0);
    display.println(F("PILOT CONTROL DEVICE"));
    display.println("");
    for (int i = 0; i < 4; i++) {
      if (i == selected) {
        display.setTextColor(SSD1306_BLACK, SSD1306_WHITE);
        display.println(options[i]);
      } else if (i != selected) {
        display.setTextColor(SSD1306_WHITE);
        display.println(options[i]);
      }
    }

// If the selected option is 0, the first option of the main menu is
  accessed, which corresponds with the "Preset shows" menú

  } else if (entered == 0) {
    down = digitalRead(3);
    up = digitalRead(2);
    enter = digitalRead(4);
    back = digitalRead(5);

    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(0, 0);
    display.println(F("PILOT CONTROL DEVICE"));
    display.println(F("PRESET SHOWS"));

    for (int i = 0; i < 4; i++) {
      if (i == selected) {
        display.setTextColor(SSD1306_BLACK, SSD1306_WHITE);
        display.println(options_shows[i]);
      } else if (i != selected) {
        display.setTextColor(SSD1306_WHITE);
        display.println(options_shows[i]);
      }
    }

// If the selected option is 1, the second option of the main menu
 is accessed, which corresponds to the LIN pilots control menu


  } else if (entered == 1) {
    down = digitalRead(3);
    up = digitalRead(2);
    enter = digitalRead(4);
    back = digitalRead(5);
```

4

```
      display.clearDisplay();
      display.setTextSize(1);
      display.setTextColor(SSD1306_WHITE);
      display.setCursor(0, 0);
      display.println(F("PILOT CONTROL DEVICE"));
      display.println(F("LIN PILOTS CONTROL"));

      for (int i = 0; i < 4; i++) {
        if (i == selected) {
          display.setTextColor(SSD1306_BLACK, SSD1306_WHITE);
          display.println(options_LIN[i]);
        } else if (i != selected) {
          display.setTextColor(SSD1306_WHITE);
          display.println(options_LIN[i]);
        }
      }
    }

// If the selected option is 2, the third option of the main menu is
    accessed, which corresponds with the ON/OFF pilots control menu

    else if (entered == 2) {
      down = digitalRead(3);
      up = digitalRead(2);
      enter = digitalRead(4);
      back = digitalRead(5);

      display.clearDisplay();
      display.setTextSize(1);
      display.setTextColor(SSD1306_WHITE);
      display.setCursor(0, 0);
      display.println(F("PILOT CONTROL DEVICE"));
      display.println(F("ON/OFF PILOTS CONTROL"));

      for (int i = 0; i < 4; i++) {
        if (i == selected) {
          display.setTextColor(SSD1306_BLACK, SSD1306_WHITE);
          display.println(options_ON_OFF[i]);
        } else if (i != selected) {
          display.setTextColor(SSD1306_WHITE);
          display.println(options_ON_OFF[i]);
        }
      }

// If the selected option is 3, the fourth option of the main menu
   is accessed, which corresponds to the credits display.

    } else if (entered == 3) {
      display.clearDisplay();
      display.setTextSize(1);
      display.setTextColor(SSD1306_WHITE);
      display.setCursor(0, 0);
      display.println(F("PILOT CONTROL DEVICE"));
      display.println(F("CREDITS"));
      display.setTextColor(SSD1306_WHITE);
      display.println(F("Eric Domek Aguila \n Monachil \n Granada \n
  Spain"));
    }

    display.display();
}
```

Listado 4.8 – *Definition of the state of the screen when each menu option is accessed*

4.3.2 VS1838 IR receiver code

In subsubsection 4.1.1.9 we decided to use a VS1838 as an IR receiver in order to control the board from distance. In this section we will detail the code that the Arduino UNO must execute to act correctly as an IR receiver and interpret the encoded commands received as commands to control the menu programmed in subsection 4.3.1.

First of all, it is mandatory to include the infrared remote header library "***IRremote***". This library contains all the necessary commands to receive the IR encoded information and decode it.

```
#include <IRremote.h>  //including infrared remote header file
```

Listado 4.9 – *Import of the necessary library*

Now we must define the digital PIN of the Arduino UNO in which we will connect the output pin of the VS1838. That is, the PIN in which the encrypted information will be received. We must also pass this PIN as an argument to a library function to let it know. Finally, we must initialize a variable that will be where the encoded information received is saved.

```
int RECV_PIN = 2; // the pin where you connect the output pin of IR
 sensor
IRrecv irrecv(RECV_PIN);
decode_results results;
```

Listado 4.10 – *Definition of the receiving digital PIN and the variable containing the coded information*

Next, as we did in the previous section, we define the setup function. In this case, we only have to initialize the serial port at 9600 bauds and enable the reception of IR commands.

```
void setup()
{
Serial.begin(9600);
irrecv.enableIRIn();
{
```

Listado 4.11 – *Setup function*

Finally we define the loop function, which is executed permanently and cyclically. In this case, all it does is wait for IR data to be available in the input buffer to display the information encoded in hexadecimal format by the serial monitor.

```
void loop()
{
if (irrecv.decode(&results))// Returns 0 if no data ready, 1 if data
  ready.
{
 int results.value = results;    // Results of decoding are stored in
  result.value
 Serial.println(" ");
 Serial.print("Code: ");
 Serial.println(results.value); //prints the value a a button press
 Serial.println(" ");
 irrecv.resume();    // Restart the ISR state machine and Receive the
  next value
}
```

Listado 4.12 – *Loop function of IR receiver*

4.3.3 DF-Player mini and VSP code

As we are already occupying the default serial port of the Arduino UNO to be able to program it externally (via USB) we need to define another serial port through software called virtual serial port if we want to control more devices. In the case of the DF-Player mini we must do it.

The first thing we must do is import both libraries so that they provide us with the control commands for the DF-Player mini and the VSP.

```
#include "AltSoftSerial.h"
#include "DFRobotDFPlayerMini.h"
```

Listado 4.13 – *Importing necessary libraries*

Now we define the pins of the Arduino UNO where we have connected the TX and RX pins of the DF-Player mini and we also let the VSP library know them. Also we create an object for the DF-Player mini.

```

static const uint8_t PIN_MP3_TX = 9; // Connects to module's RX
static const uint8_t PIN_MP3_RX = 8; // Connects to module's TX
```

```
AltSoftSerial softwareSerial(PIN_MP3_RX, PIN_MP3_TX);

// Create the Player object
DFRobotDFPlayerMini player;
```

Listado 4.14 – *Importing necessary libraries*

We initialize the VSP corresponding to the DF-Player mini, set the volume to 50 and show a message on the screen if the connection has been successful and another if it has not.

```
softwareSerial.begin(9600);

if (player.begin(softwareSerial)) {
    Serial.println("OK dfplayer");

    // Set volume to maximum (0 to 30).
    player.volume(50);
} else {
    Serial.println("Connecting to DFPlayer Mini failed!");
}
```

Listado 4.15 – *Importing necessary libraries*

Once we have executed all the previous commands, all that remains is to include the playback command in the conditions that we want each song to play.

```
player.play(\\Number of the song inside microSD card here)
```

Listado 4.16 – *Importing necessary libraries*

4.3.4 74HC595 shift registers code

In Figure 4.21 we decided to use 4 74HC595 shift register to implement the necessary output digital PINs that we need to control each pilot function separately. This section will explain the code that we must implement in the arduino to control these 4 shift registers and make possible the individual control of each of its output pins.

For this case we have decided to use a library created manually by a Zurich user [45]. First, we import this library.

```
#include <ShiftRegister74HC595.h>
```

Listado 4.17 – *Inclusion of shift register library*

Next we must create a global object to which we must introduce 4 parameters:

- Number of shift registers dasy chained.
- Data PIN.
- Clock PIN.
- Latch PIN

In our case, we have a total of 4 dasy chained shift registers, the data pin is 9, the clock pin is 11, and the latch pin is 12.

```
// Create a global shift register object
// Parameters: <number of shift registers> (data pin, clock pin,
 latch pin)
ShiftRegister74HC595<4> sr(9, 11, 12);
```

Listado 4.18 – *Global shift register object definition*

In this case, it is not necessary to declare any initialization command inside the setup function, so it can be left empty.

```
void setup{
}
```

Listado 4.19 – *Setup function of shift registers code*

Below we show some examples of very useful functions that we can implement once we have the 4 shift registers connected and the library initialized and configured.

- We can activate or deactivate all the output pins **at the same time** with the functions "*setAllHigh*" or "*setAllLow*" respectively.
- One of the most important functions is that we can **separately activate or deactivate** each digital PIN of the 32 that we have in total at the exit with the function "*set*".
- We can define the LOW or HIGH state of all the pins at the same time with the "*setAll*" function.
- We can even read the status of a PIN using the "*get*" function.
- We can store the future state of each PIN without changing it at that moment and choose the moment in which to update all the changes made with the "*setNoUpdate*" and "*updateRegisters*" functions.

```
void loop() {

  // setting all pins at the same time to either HIGH or LOW
  sr.setAllHigh(); // set all pins HIGH
  delay(500);
  sr.setAllLow(); // set all pins LOW
  delay(500);


  // setting single pins
  for (int i = 0; i < 8; i++) {

    sr.set(i, HIGH); // set single pin HIGH
    delay(250);
  }


  // set all pins at once
  uint8_t pinValues[] = { B10101010 };
  sr.setAll(pinValues);
  delay(1000);


  // read pin (zero based, i.e. 6th pin)
  uint8_t stateOfPin5 = sr.get(5);
  sr.set(6, stateOfPin5);


  // set pins without immediate update
  sr.setNoUpdate(0, HIGH);
  sr.setNoUpdate(1, LOW);
  // at this point of time, pin 0 and 1 did not change yet
  sr.updateRegisters(); // update the pins to the set values
}
```

Listado 4.20 – *Setup function of shift registers code*

4.3.5 LIN transceiver code

Let us remember that the LIN protocol has a slave-master topology (more information in Appendix A). In this case, our product would be the communication master and the 2018 Audi Q8 drivers would be the slaves.

The Arduino UNO code that we must implement in order to carry out the control of the pilots must act as "*Exxotest Muxdiag-II*". That includes generating the necessary control frames and responding to itself with the necessary data field to activate the desired function and generating the headers of the defect diagnosis frames too as explained in paragraph 3.4.3.2.1.

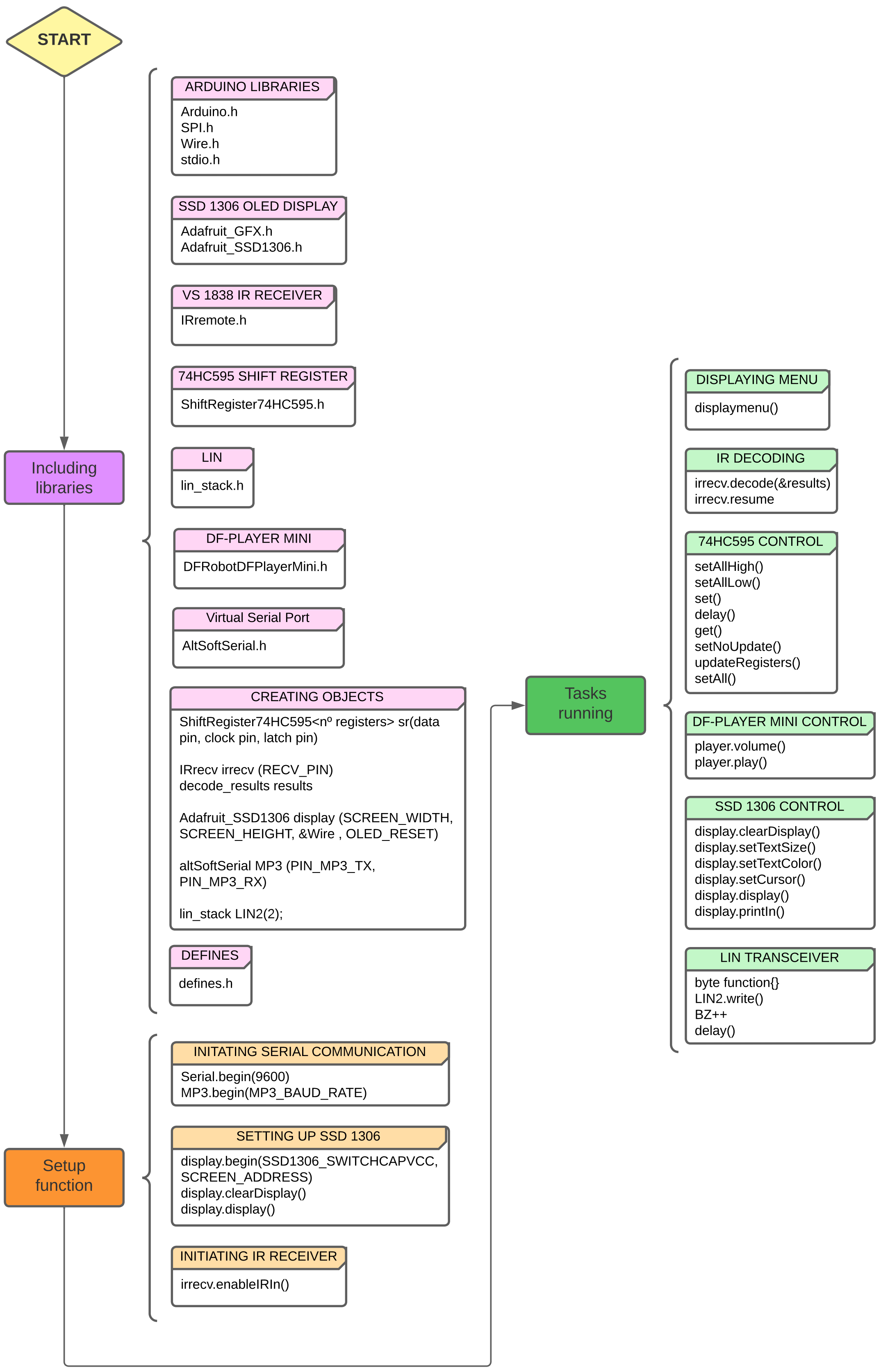
The rest of the work is done by the pilots: receiving the control frames, activating the corresponding functions and writing in the data field of the diagnostic frames the necessary information to let the master know about possible failures.

In order to implement this code we have resorted to an external library created by a user [5] which offers the user the possibility of building an array of bytes, which is the length of the message and contains packages to be send to the module. An example of code that we could use to implement the LIN schedule is the following. In it, the master (our product) follows the routine seen in Figure 3.58 to specifically activate the dynamic tail function. To activate other functions, simply it would be necessary to change the value of the signals in the data field, since the schedule followed is always the same.

```
#include <lin_stack.h>

lin_stack LIN2(2); // Creating LIN Stack objects, 2 - second channel

void setup() {
  // Nothing to do here.
}

void loop() {
  // Example of activating dynamic tail ON function

  while(){

  byte dynamic_ON[8] = {0x00, 0x01 << 8 | BZ, 0x00, 0x00, 0x00, 0x10
 , 0x0E, 0x00} // Creation of the control frame data field, the BZ
 signal present in the 4 least significant bits of byte 1 is
 incremented each time the frame is sent.

  LIN2.write(0x0A, dynamic_ON, 8);    //Dynamic ON control frame
  BZ ++;    // BZ signal increased
  delay(10);    // Interframe time

  LIN2.write(0x0B, 0x00, 8);    // Defect diagnosis frame for 0x0B
  delay(10); // Interframe time

  byte dynamic_ON[8] = {0x00, 0x01 << 8 | BZ, 0x00, 0x00, 0x00, 0x10
 , 0x0E, 0x00};
  LIN2.write(0x0A, dynamic_ON, 8);    //Dynamic ON control frame
  BZ ++;    // BZ signal increased
  delay(10); // Interframe time

  LIN2.write(0x0C, 0x00, 8);    // Defect diagnosis frame for 0x0C
  delay(10); // Interframe time

  byte dynamic_ON[8] = {0x00, 0x01 << 8 | BZ, 0x00, 0x00, 0x00, 0x10
 , 0x0E, 0x00};
  LIN2.write(0x0A, dynamic_ON, 8);    //Dynamic ON control frame
  BZ ++;    // BZ signal increased
  delay(10); // Interframe time

  LIN2.write(0x0D, 0x00, 8);    // Defect diagnosis frame for 0x0D
  delay(10); // Interframe time

  byte dynamic_ON[8] = {0x00, 0x01 << 8 | BZ, 0x00, 0x00, 0x00, 0x10
 , 0x0E, 0x00};
  LIN2.write(0x0A, dynamic_ON, 8);    //Dynamic ON control frame
  BZ ++;    // BZ signal increased
  delay(10); // Interframe time
```

```
    LIN2.write(0x0E, 0x00, 8);    // Defect diagnosis frame for 0x0E
    delay(10); // Interframe time

    }


}
```

Listado 4.21 – *Example of LIN code for turning ON dynamic tail*


START
Including libraries
ARDUINO LIBRARIES
Arduino.h
SPI.h
Wire.h
stdio.h
SSD 1306 OLED DISPLAY
Adafruit_GFX.h
Adafruit_SSD1306.h
VS 1838 IR RECEIVER
IRremote.h
74HC595 SHIFT REGISTER
ShiftRegister74HC595.h
LIN
lin_stack.h
DF-PLAYER MINI
DFRobotDFPlayerMini.h
Virtual Serial Port
AltSoftSerial.h
CREATING OBJECTS
ShiftRegister74HC595<nº registers> sr(data pin, clock pin, latch pin)
IRrecv irrecv (RECV_PIN)
decode_results results
Adafruit_SSD1306 display (SCREEN_WIDTH, SCREEN_HEIGHT, &Wire , OLED_RESET)
altSoftSerial MP3 (PIN_MP3_TX, PIN_MP3_RX)
lin_stack LIN2(2);
DEFINES
defines.h
Setup function
INITATING SERIAL COMMUNICATION
Serial.begin(9600)
MP3.begin(MP3_BAUD_RATE)
SETTING UP SSD 1306
display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)
display.clearDisplay()
display.display()
INITIATING IR RECEIVER
irrecv.enableIRIn()
Tasks running
DISPLAYING MENU
displaymenu()
IR DECODING
irrecv.decode(&results)
irrecv.resume
74HC595 CONTROL
setAllHigh()
setAllLow()
set()
delay()
get()
setNoUpdate()
updateRegisters()
setAll()
DF-PLAYER MINI CONTROL
player.volume()
player.play()
SSD 1306 CONTROL
display.clearDisplay()
display.setTextSize()
display.setTextColor()
display.setCursor()
display.display()
display.println()
LIN TRANSCEIVER
byte function{}
LIN2.write()
BZ++
delay()

Chapter 5

Integration, test and verification

Once the entire process of **analysis** and **design** of our product has been carried out in chapter 3 and chapter 4, this chapter shows the manufacturing process of the product fully documented with multimedia content. In addition, once the assembly of the product is finished, a series of tests are carried out on it to verify the correct operation of all the modules and verify that no serious errors have been made both in the design and in the assembly of the PCB.

As we mentioned in subsubsection 4.1.2.1, given the short time available in the final stretch of the project, we decided to build the PCB in two different ways: ordering it on the Chinese website `jlcpcb.com` and manufacturing it in GranaSAT's own laboratory. In this way, if the manufacturing in the laboratory failed, we would always be assured that the PCB manufactured in China with much more advanced machinery would arrive at some point. In fact, due to time constraints, the PCB could not finally be manufactured in the laboratory itself.

5.1 PCB manufactured by `jlcpcb.com` assembly

Once the 5 PCBs that we ordered from the Chinese manufacturer arrived at the laboratory, we were ready to start the soldering process of all the components chosen in the previous chapters and purchased from different manufacturers. Figure 5.1 shows a view of the TOP and BOTTOM layers of the unsoldered PCB, where we can notice that the final result is completely faithful to the model designed in Altium Designer® 21.

Let us remember that this PCB has components of different assembly technology: some are SMD and others are THT. To solder the PCB we must start by soldering the SMD components, since, if we solder the THT components first, the relief they represent will not allow the next step to be carried out correctly: **the application of the solder paste using a Stencil.**

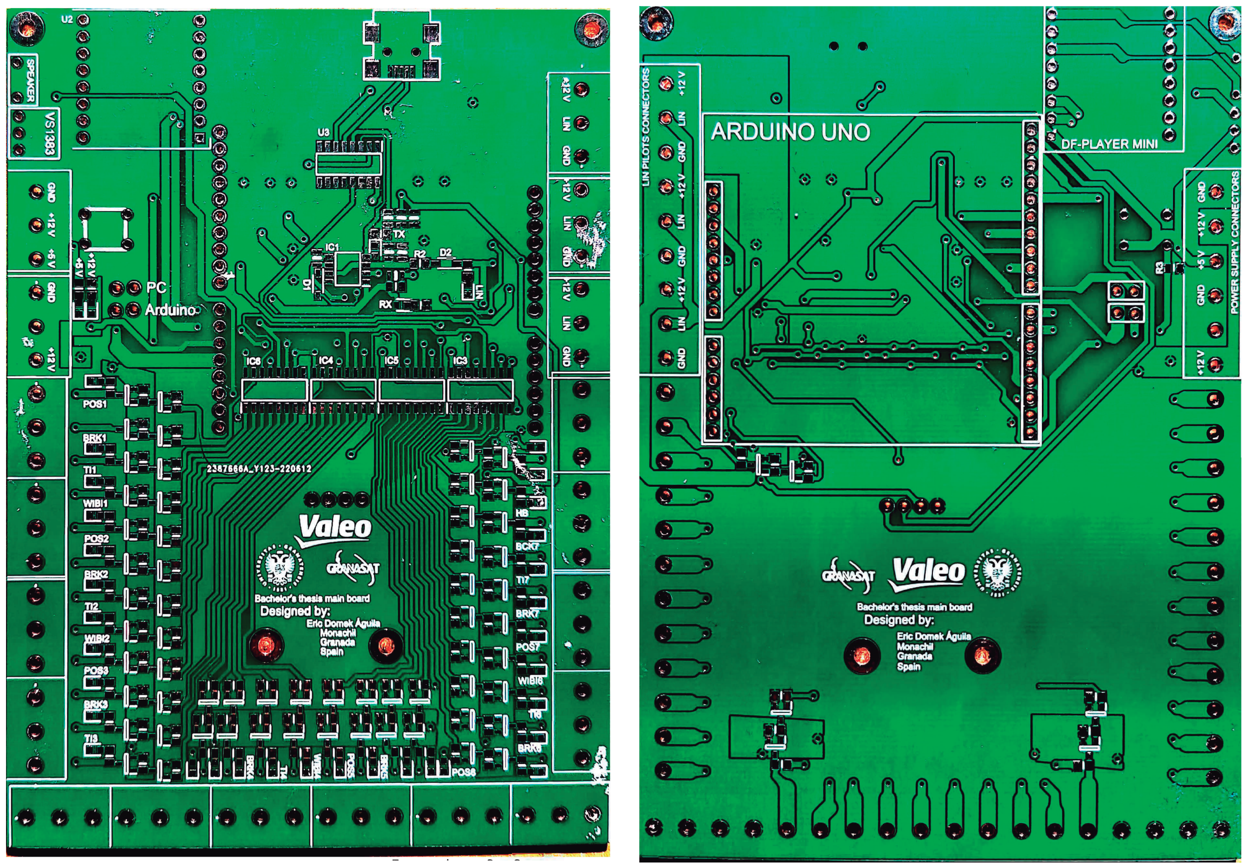
5.1.1 SMD placing: PCB fixing, stencil placement and solder paste

In principle, we have two options to solder the SMD components to our board:

- **By hand**: solder each SMD component separately using a soldering iron. This method is not at all adequate given the size of some of the components that the board mounts. In addition to the inaccuracies that it would entail, the time it requires is totally disproportionate since there are a huge number of components.

- **Using Stencil and solder paste:** The Stencil is placed on the PCB so that all the PADs are correctly aligned and the solder paste is deposited. The placing of the components is carried out in an approximate manner and the PCB is introduced into the oven. A much more precise, fast and simple method than the previous one, that is why we will carry it out.

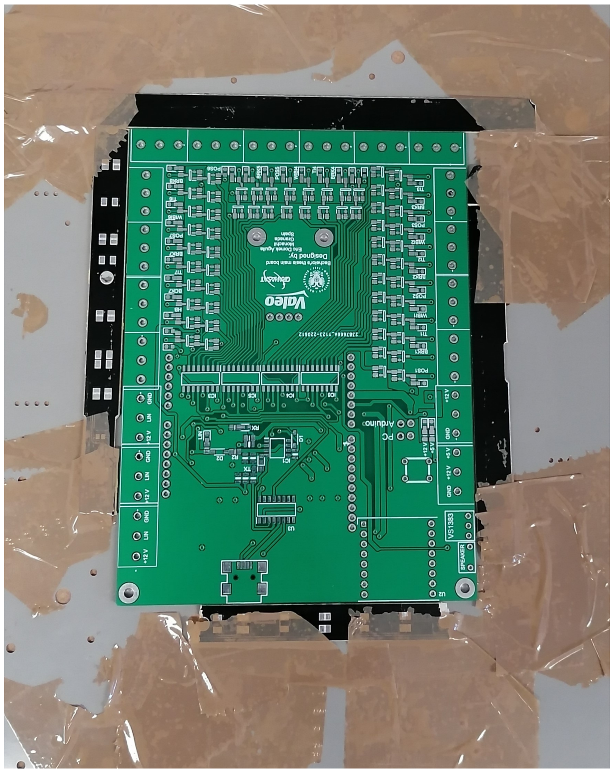
The first step that we must follow to solder the PCB using solder paste and Stencil is to fix and immobilize both the plate and the Stencil so that its indentations coincide perfectly with the pads on the plate. One way to show that we have done it right is to notice that absolutely nothing green is visible through the Stencil holes, but we can only see the characteristic metallic gray of the PADs.



(a) *TOP* **(b)** *BOTTOM*

Figure 5.1 – *Scanned unsoldered PCB TOP and BOTTOM*

In Figure 5.2 (a) we can see the end of the plate fixing process. For this we have used remnants of an old PCB of the same thickness (1.6 mm) and we have placed them with adhesive tape on the edges of ours. In this way our plate is fixed without having to stick it directly with adhesive tape. This saves us from having to deal with the difference in thickness that it would entail when applying the solder paste, which would mean that there would be more quantity in some parts than in others.


(a) *PCB fixed*

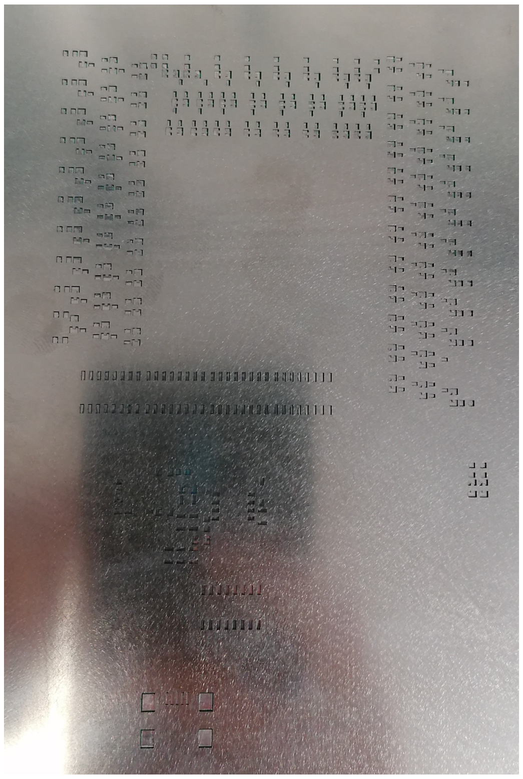
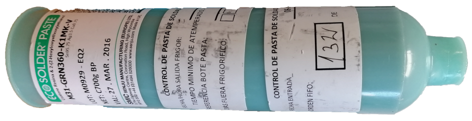

(b) *Positioned Stencil*

Figure 5.2 – *PCB Stencil application*

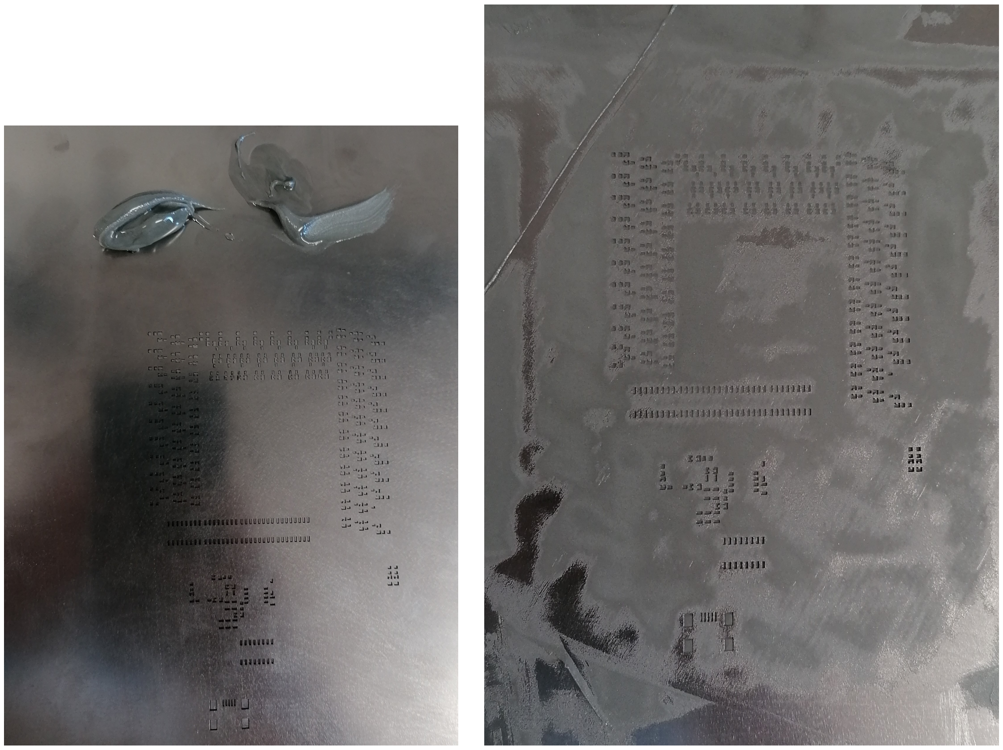
In In Figure 5.2 (b) we can see the correct way in which the Stencil should be placed so that there is no problem when depositing the solder paste. Once we have our plate and Stencil immobilized in the correct position we can proceed to apply the solder paste on the Stencil.

In Figure 5.3 we can see the solder paste that we have used in our case. To be able to apply it, we must insert the container into a gun that will press on the back and make the paste come out of the front in a controlled manner.


(a) *Outside gun*

(b) *Inside gun*

Figure 5.3 – *Used solder paste bottle*



(a) *Without spreading* **(b)** *Spread*

Figure 5.4 – *Solder paste applied in Stencil*

As we can see in Figure 5.4 (a), first we deposit a little solder paste next to the holes of the Stencil. What we must do next is spread the solder paste with the help of a spatula and make sure that it is inserted through each and every one of the holes evenly. Once we have done that, we get the result of Figure 5.4 (b).

In Figure 5.6 we can see the appearance of the PCB after removing the Stencil. As we can see, the solder paste is deposited precisely on the solder pads, which is where we are interested. Once this is done, we are ready to proceed with the placing of the components and the baking.

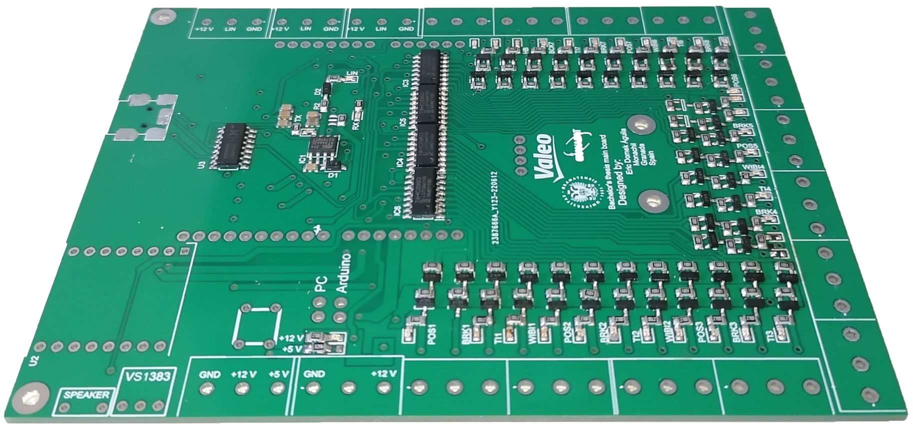
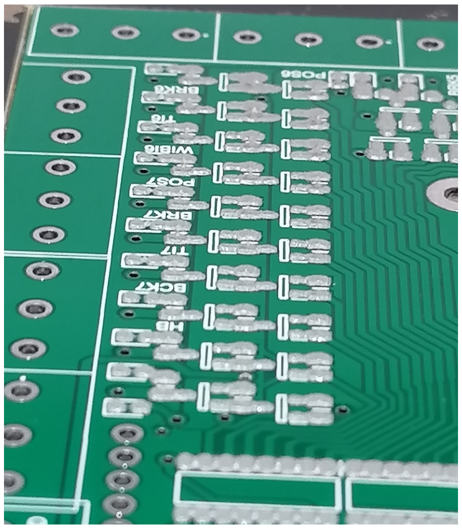
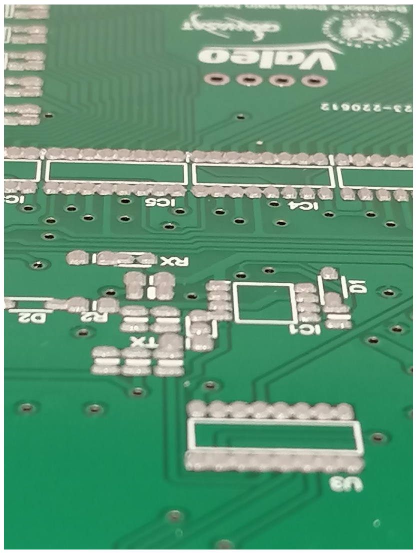


Figure 5.5 – *PCB with SMD components soldered*


(a) *View 1*


(b) *View 2*

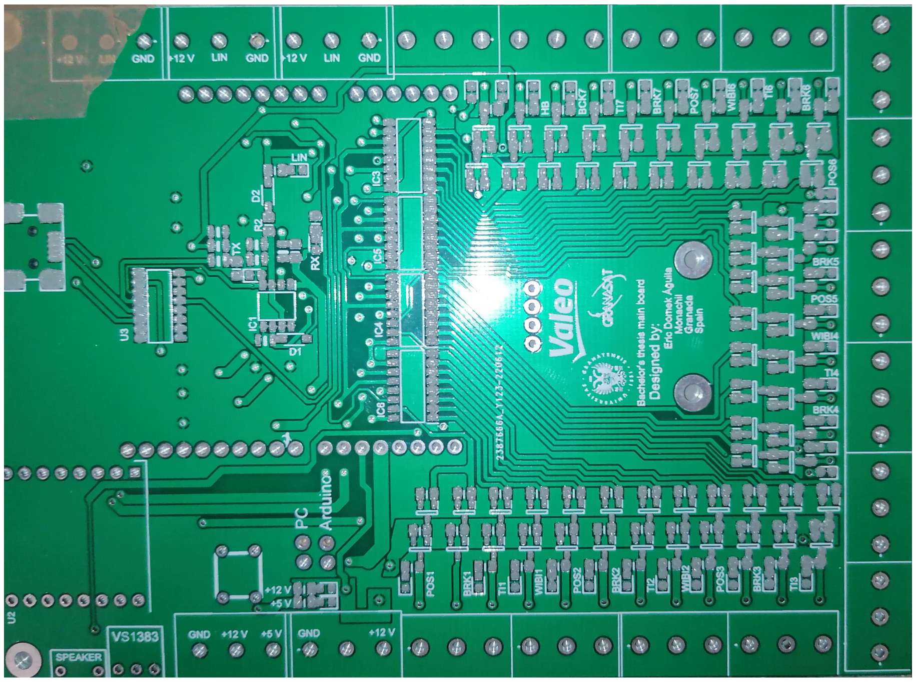

(c) *TOP view*

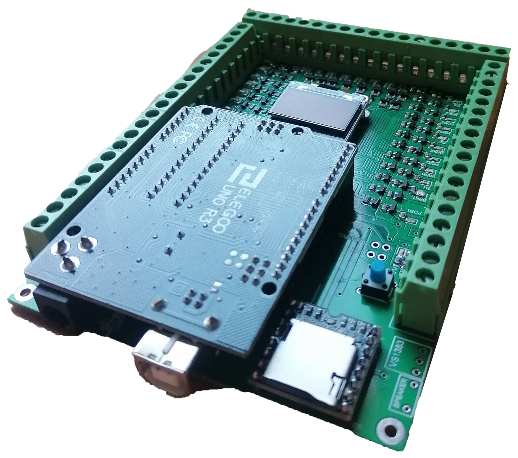
Figure 5.6 – *PCB after applying solder paste*

Figure 5.5 shows the appearance of the board after finishing soldering all the SMD components on it. We have decided to solder the BOTTOM layer by hand (with a soldering iron), since it is impossible to turn the PCB over once it has solder paste on one of the layers. However, this is not too much of a problem for us, since there are not many components on the BOTTOM.

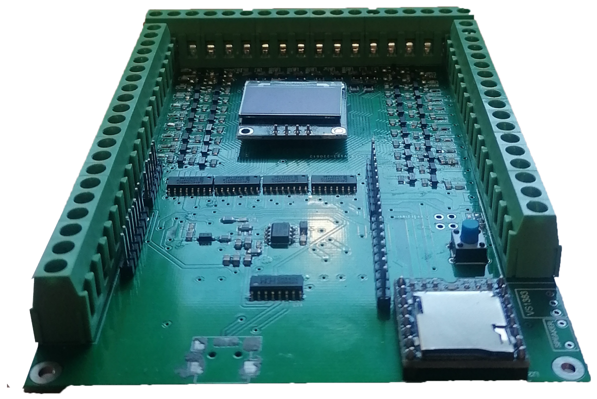
5.1.2 THT components placing

Once we are done with placing the surface mount components, we can continue soldering the THT components to the board. This part of the plate assembly is much simpler than the previous one. To solder a THT component, all we have to do is fit its pins into the mounting holes and apply a small drop of solder to the opposite side of the PCB. In this way the component is mechanically fixed and electrically connected.

In Figure 5.7 we can see the final appearance of the PCB with the THT components soldered (except the VS1838, which was soldered after taking the picture). We can notice that the final appearance of the product is extremely faithful to the appearance of the 3D model generated with the Altium Designer® 21 software. This is a demonstration of the level of precision achievable in any design by using the right tools and knowledge.



(a) *With Arduino UNO*



(b) *Without Arduino UNO*

Figure 5.7 – *PCB with THT compopnents soldered*

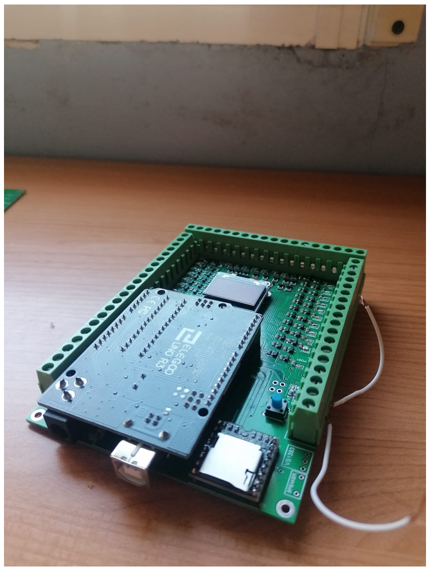
5.1.3 Modules testing

In this section we will carry out a series of tests on the finished PCB to check the correct functioning of the modules. Videos are included as support for the demonstration.

5.1.3.1 IR receiver, OLED display and MP3 player

The operation of the modules that offer a friendly user experience is independent of the rest of the modules that offer the true functionality for which the product has been designed, which is why we will carry out an independent test of these.

As we can see in video 5.1, a code has been implemented that makes the DF-Player mini reproduce, through the speaker connected to the corresponding terminals, different MP3 audio files stored on a microSD card. In addition, through the IR remote control, we can choose each of the 7 existing audio tracks by pressing the corresponding number. Finally, the SSD1306 OLED screen shows the name of the song that is playing at all times.

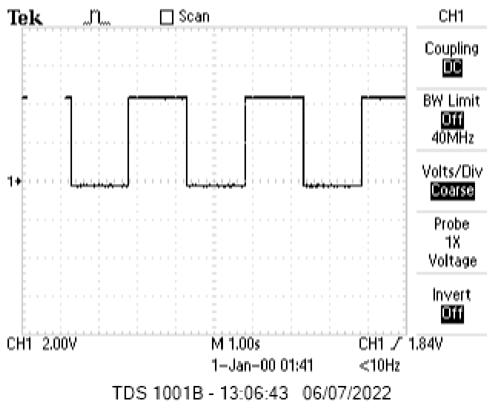


Video 5.1 – *Testing of SSD1306, VS1838 and DF-Player mini*

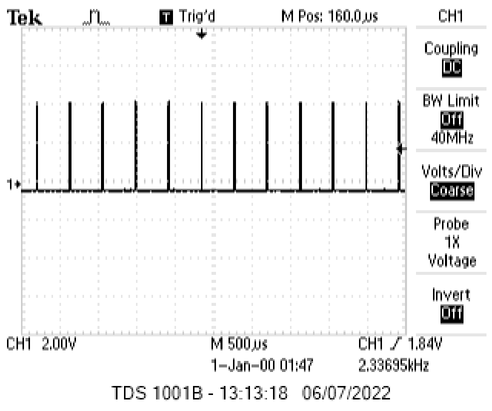
5.1.3.2 Shift register module testing

To check the correct operation of the shift register module whose function is to increase the number of digital input/output pins of the Arduino UNO, we will first check that all the outgoing signals of the microcontroller are arriving at the correct pins of the shift register. For this we will use an oscilloscope with which we must check the existence of 4 signals:

- **Serial data input**: This signal sends the 4 bytes that represents the state of each digital output depending of the loaded code. In Figure 5.8 we can see two examples of functioning:
  - (a) Represents turning every output ON and OFF every 1.5 seconds, so the 4 bytes are all ones and all zeros every half-period.
  - (b) Represents only one output ON, so only one of the 32 bits is '1' and the rest are '0'.
- **Latch clock:** This signal produces a rising edge every time something changes in the bytes that make up the serial data. In the Figure 5.9 (a) we can see an example of how it looks when the serial data corresponds to that of the Figure 5.8 (a). We can see that, since there are only two changes (all ON or all OFF), the signal only has two pulses that are repeated periodically.
- **Shift clock:** This is a clock signal where each rising edge causes the data present at the serial data input to be loaded into the shift registers. This signal is periodic and always has the same shape as we can see in the Figure 5.9 (b).
- **Output Enable:** For all shift registers to work correctly, this signal must be low, as we see in Figure 5.10

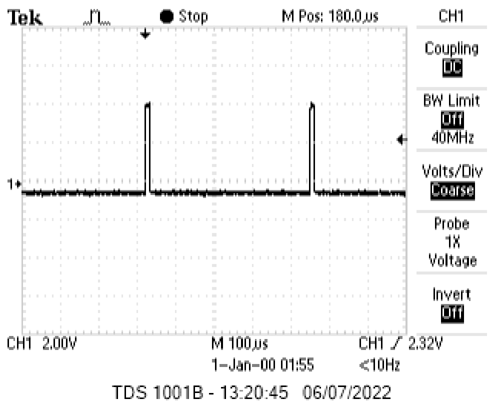


(a) *Pattern 1*

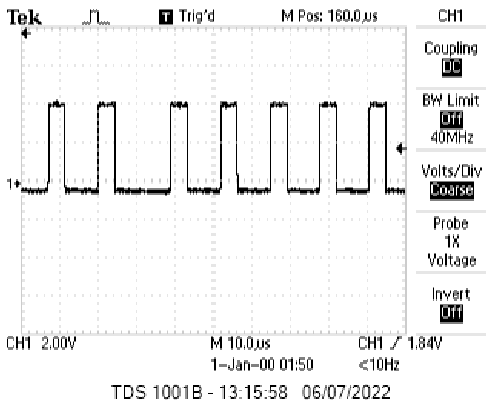


(b) *Pattern 2*

Figure 5.8 – *Serial data testing patterns*



(a) *Latch clock*



(b) *Shift clock*

Figure 5.9 – *Shift and latch clock signal testing*

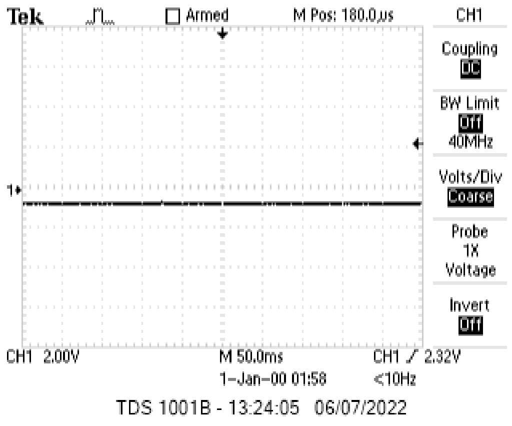


Figure 5.10 – *Output enable signal testing*

The next thing we should check is that, indeed, when we activate certain outputs in our Arduino UNO code, they have a voltage of +5 V. The first time this check was made, everything worked perfectly. However, for some unknown reason, the successive tests that were carried out did not give good results, since outputs are activated and deactivated randomly. It is intuited that it must be a failure of the integrated circuits themselves, since all the signals arrive correctly at them, as we have just verified with the oscilloscope.

5.1.3.3 Wooden plank with headlights turned ON

In the Figure 5.11 we can see a demonstration of all the functions of all the headlights turned ON at the same time to show that they all work correctly.

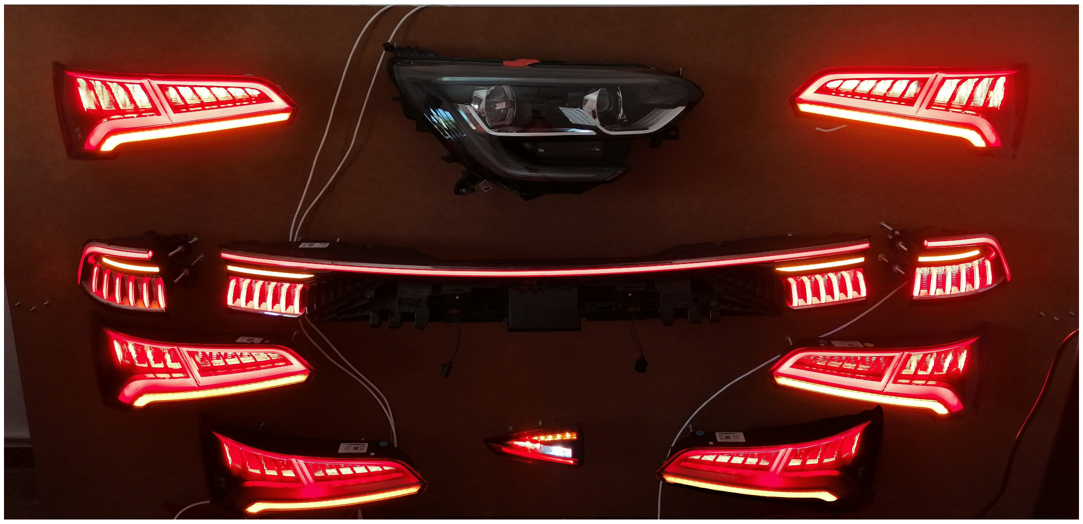


Figure 5.11 – *Wooden plank with all headlights turned ON*

Chapter 6

Conclusions and future lines

This document has shown the tough procedure of performing a reverse engineering process to an existing product. To do such thing, we have presented the development of a demonstrator device for pilot/headlights corresponding to high-end vehicles from prototype to final design through the different stages that entails.

We have managed to understand and familiarized with the process of a system design in the engineering industry. This Bachelor's Thesis has allowed a telecommunication engineering student, as the author, to better understand the concepts learned during the academic years of electronic design. During the development of this work, especially at the beginning of the project, we realized that the student's base knowledge was not enough to tackle this project. Although he knew the basic notions learned during the career, the lack of experience made the first phases of the project slower. In the course of the academic period, there has not been so much emphasis on bringing the student closer to what the industry really is as much as the theoretical knowledge, therefore, the student does not know well how to function when designing a product or even getting along at the laboratory. Throughout this time period we have learned a little about how this industry works, its standards and how important a good documentation is. We have also learn how to work in a science lab with other partners, where everyone can learn from each other even though the project's they are working in are different.

The pilot demonstrator device is yet far from being ready to be sold since we only did the first prototype from scratch. However, we think we have lain the groundwork for this purpose.

The lines we need to work on in the future to get this product ready to be sold are:

- Carry out all the remaining tests on the product, which would be:
  - **LIN transceiver module:** Check that both the code developed for it and the electronic scheme are capable of acting as the "*Exxotest USB-MUXDIAG-II*" and controlling the set of LIN pilots of the Audi Q8 2018.
  - **ON/OFF pilots control:** Although this module came to work, its operation was not documented through multimedia content, so it cannot appear in this report. A future goal would be to fix the problem that caused it to fail and retest. It is almost certainly believed that replacing all 4 shift registers would be the solution.
- **Develop the remaining code** to make the pilots respond to the rhythm of the music that is playing through the DF-Player.
- **Optimize the PCB design:** it is the first serious PCB design project that the student carries out, so, although the design is satisfactory, more elements could be included and the existing ones optimized to

achieve a higher quality product. For example, the **width of the tracks** could be further optimized, a way could be found to **redistribute the terminal blocks** in order to make the PCB smaller and take more advantage of the space, since the current one has some areas with low component density.

- The necessary code could be developed so that our product is capable of **interpreting the defect diagnosis frames** whose signals present in the data field are written by the slave pilots. In this way, our product would no longer have only demonstrative purposes, but we would be talking about a powerful error diagnosis tool that could be used in regular debugging processes in the automotive industry.
- A mobile application could be developed to be able to select all the control/diagnostic options in a comfortable way without having to use the OLED display and the IR remote controller, which would save space on the board and give it much more accessibility and professionalism. This option would require the inclusion of a Wi-Fi or Bluetooth module.
- To provide the product with greater realism and professionalism, it would be ideal to r**eplace the current microcontroller** (Arduino UNO), which is usually used as a platform for testing prototypes and is quite limited, with a generic microcontroller in format, which would provide the product with greater freedom to be configured.

We would like to conclude by saying that it has been quite an enriching experience developing this project, and we did not expect us to have this ability to adapt to the problems and difficulties encountered along a project of this type. Once again, the future is exciting and we hope to work on perfect this design until it is ready to be sold.

Bibliography

[1] ARDUINO.CC. Arduino official website https://www.arduino.cc/.

[2] ATMEL. *Datasheet of ATA6662C*, 2014. http://ww1.microchip.com/downloads/en/devicedoc/atmel-4916-lin-networking-ata6662c_datasheet.pdf.

[3] DEEPTI BANKAPUR. What are teardrops? https://resources.pcb.cadence.com/blog/pcb101-include-teardrops-in-your-designs-to-save-your-tears-later.

[4] DSF.MY. Picture of Mercedes-Benz headlights evolution https://www.dsf.my/2019/12/mb-lighting-evolution/.

[5] DURONFLO. Arduino UNO library for implementing slave-master LIN topology github.com/macchina/LIN.

[6] EBAY.ES. Infiniti Q30 2015 Trunk LHD/RHD cost https://www.ebay.es/itm/264589965942.

[7] EMEDEC.COM. Standard MDF wooden planks dimensions https://www.emedec.com/medidas-tableros-descubre-cuales-son-los-estandares/.

[8] ES.ALIEXPRESS.COM. AMUX-2C2L adapter cost https://es.aliexpress.com/item/1005003327730071.html?

[9] ES.ALIEXPRESS.COM. DB9 to 3 pin adapter cost https://es.aliexpress.com/item/1005001320859431.html.

[10] ES.ALIEXPRESS.COM. ATA6662C LIN transceiver cost https://es.aliexpress.com/item/1005002274241712.html?

[11] ES.ALIEXPRESS.COM. USB mini B port cost https://es.aliexpress.com/item/1005003394774696.html?

[12] ES.ALIEXPRESS.COM. CH340C cost https://es.aliexpress.com/item/32877415119.html?

[13] ES.ALIEXPRESS.COM. 74HC595 cost https://es.aliexpress.com/item/1005002274578264.html?

[14] ES.ALIEXPRESS.COM. DF-Player mini cost https://es.aliexpress.com/item/1005001474899783.html?

[15] ES.ALIEXPRESS.COM. FDN342P P-Channel MOSFET cost https://es.aliexpress.com/item/4000144049197.html?

[16] ES.ALIEXPRESS.COM. 2N7002 N-Channel MOSFET cost https://es.aliexpress.com/item/1005003667253182.html?

[17] ES.ALIEXPRESS.COM. VS1838 IR receiver cost https://es.aliexpress.com/item/32848590741.html?

[18] ES.ALIEXPRESS.COM. Phoenix Contact Terminal Blocks cost https://es.aliexpress.com/item/1005002950641585.html?

[19] ETC2. *Datasheet of VS1838*. blob:https://pdf1.alldatasheet.es/1b36c072-e1c7-4cfe-8d2b-fcf1de1f2970.

[20] EXXOTEST. *USB-MUXDIAG-II communication interface user guide*, 2012. https://exxotest.com/wp-content/uploads/2018/01/en_user_guide_USB-MUXDIAG-II.pdf.

[21] FALCH, M. Lin bus explained - a simple intro. https://www.csselectronics.com/pages/lin-bus-protocol-intro-basics Accessed may 2022.

[22] FALCH, M. *CAN bus - the ultimate guide*, 2022. CSS Electronics.

[23] HTDISPLAY.COM. *TFT LCD display HT0280CI01BR1 information*. http://en.htdisplay.com/product/219.html.

[24] ILITEK. *Datasheet of ILI9341 TFT LCD display*. blob:https://pdf1.alldatasheet.com/a5124214-6863-4f9d-8c93-7ba6163d5bb7.

[25] JLCPCB.COM. Chinese PCB manufacturer jlcpcb.com.

[26] KVASER. Introduction to the lin bus. https://www.kvaser.com/about-can/can-standards/linbus/ Accessed may 2022.

[27] LABORATORY, C. U. V. E. Basic description of LIN (local interconnect network). https://cecas.clemson.edu/cvel/auto/Buses/LIN.html Accessed may 2022.

[28] LAURENCE WILLIAMS. Differences between Microprocessor and Microcontroller https://www.guru99.com/difference-between-microprocessor-and-microcontroller.html.

[29] LIFEWIRE.COM. Pinout of ATX power supply 24 PIN connector https://www.lifewire.com/atx-24-pin-12v-power-supply-pinout-2624578.

[30] LUCIDCHART.COM. Web used for making flux and block diagrams evolution https://www.lucidchart.com//.

[31] MICROCHIP.COM. Structure of LIN description file. https://microchipdeveloper.com/lin:protocol-app-ldf Accessed may 2022.

[32] MICROCHIP.COM. Structure of the LIN frame message. https://microchipdeveloper.com/lin:protocol-dll-lin-message-frame Accessed may 2022.

[33] MOTOCOCHE.COM. Audi Q5 2020 Trunk Left/Right Light cost https://motocoche.com/recambios/piloto-trasero-derecho-audi-q5-fyb-20-tdi-2/.

[34] MOUSER.ES. *Datasheet of blue LED*, 2021. https://www.mouser.es/datasheet/2/445/150060BS75003-2907472.pdf.

[35] MOUSER.ES. *Datasheet of red LED*, 2021. https://www.mouser.es/datasheet/2/445/150060SS75003-2907500.pdf.

[36] PHOENIXCONTACT.COM. Phoenix Conctact official website https://www.phoenixcontact.com/es-es/.

[37] RECTIFIER, I. *Datasheet of IRLML6402TRPBF*. blob:https://pdf1.alldatasheet.com/0d9b811d-e4c0-472b-a120-e618be80c630.

[38] RESEARCHGATE.NET. Switched power supply blocks diagram https://www.researchgate.net/figure/Basic-switched-mode-power-supply-block-diagram-1_fig1_251910649.

[39] Saleae.com. *Datasheet of Saleae Logic 8 USB Logic Analyzer*, 2018. https://520354028-files.gitbook.io/~/files/v0/b/gitbook-legacy-files/o/assets%2F-LIrtSD7SNp69UxQ-5QC%2F-LNmCWBVt9xzKd0JvS4e%2F-LNmCi9Ud2oir7O_pM_B%2FLogic%208%20Data%20Sheet.pdf?alt=media&token=4acd44bb-eb29-4c47-8f23-61bba5993d44.

[40] Semiconductor, F. *Datasheet of P-channel MOSFET FDN342P*. blob:https://pdf1.alldatasheet.com/df95f94e-1d92-4315-91c5-82688a884568.

[41] Semiconductor, F. *Datasheet of P-channel MOSFET FDN360P*. https://pdf1.alldatasheet.com/datasheet-pdf/view/51389/FAIRCHILD/FDN360P/+_1_8-UwYhRDpKxHelKUwGzb+/datasheet.pdf#pdfjs.action=download.

[42] Semiconductor, O. *Datasheet of 74HC595*, 2000. blob:https://pdf1.alldatasheet.com/ec091aa9-a9d9-4c93-a77a-500f2577bec4.

[43] sparks.gogo.co.nz. Web for downloading CH340C drivers https://sparks.gogo.co.nz.

[44] SYSTECH, S. *Datasheet of SSD1306*, 2008. blob:https://pdf1.alldatasheet.com/2274955e-44d8-48fe-bb8a-d325c353851a.

[45] Timo Denk. Customized library for 74HC595 https://timodenk.com/blog/shift-register-arduino-library/.

[46] valeo.com. Valeo official website www.valeo.com.

[47] wch.cn. *Datasheet of CH340C*. https://www.mpja.com/download/35227cpdata.pdf.

[48] www.4pcb.com. PCB minimum tracks width calculator https://www.4pcb.com/trace-width-calculator.html.

[49] www.amazon.es. USB-MUXDIAG-II cost https://www.amazon.es/AZDelivery-Logic-Analyzer-compatible-versi%C3%B3n/dp/B01MUFRHQ2.

[50] www.amazon.es. Arduino UNO cost https://www.amazon.es/Arduino-UNO-A000066-microcontrolador-ATmega328/dp/B008GRTSV6.

[51] www.hotmcu.com. SSD1306 OLED display cost https://www.hotmcu.com/ssd1306-096-128%C3%9764-oled-display-%E2%80%93-i2cspi-interface-p-144.html?

[52] www.pccomponentes.com. ATX switching power supply cost https://www.pccomponentes.com/nox-urano-vx-750w-80-bronze-140mm-pwm.

[53] www.tarostrade.es. Renault Megane 2016 Headlight cost https://www.tarostrade.es/faro-delantero-138254/po/renault-megane-sedan-2016-228845.

[54] www.tarostrade.es. Audi Q8 2018 Fender Left/Right Lights cost https://www.tarostrade.es/luz-trasera-253269/po/audi-q8-2018-289603.

[55] www.tarostrade.es. Audi Q8 2018 Trunk Light cost https://www.tarostrade.es/luz-trasera-295971/po/audi-q8-2018-443372.

[56] www.vtsonline.co.uk. USB-MUXDIAG-II cost https://www.vtsonline.co.uk/products/p/exxotest-usb-muxdiag-ii.

Appendix A

LIN bus and protocol

A.1 Introduction

In this section we will focus on explain in depth what the LIN bus and protocol consist of, finding a way to find out how the communication between a PC and the set of LIN-controlled pilots that Valeo has provided us is carried out and obtain every singe frame corresponding to the activation of each function of each pilot.

A.2 LIN bus

LIN (Local Interconnect Network) is a bus mostly used by the automotive industry for communication between components in vehicles. It's a supplement to CAN bus.

A.2.1 Differences and comparison with CAN bus

LIN bus offers lower performance and reliability than CAN bus, but also much lower costs. The table below A.1 show some differences between both buses.

LIN	**CAN**
Lower cost (less harness and cheap nodes)	Higher cost
Single Wire 12 V	Twisted shielded dual wires 5 V
Deterministic, no bus arbitration	Lossless bitwise [1] arbitration method of contention resolution
Single master	Multiple masters
6 bits identifiers	11 or 29 bit identifiers
Up to 20 kbits/s	Up to 1 Mbit/s

Table A.1 – *Comparison between CAN and LIN buses [21] [22]*

As we have said, the CAN and LIN buses are not at the same level within the connection topology of the vehicle, but the LIN bus is a slave of the CAN bus. In the image A.1 we can see an example that illustrates the use of this topology

[1]Bitwise: Bit by bit operation

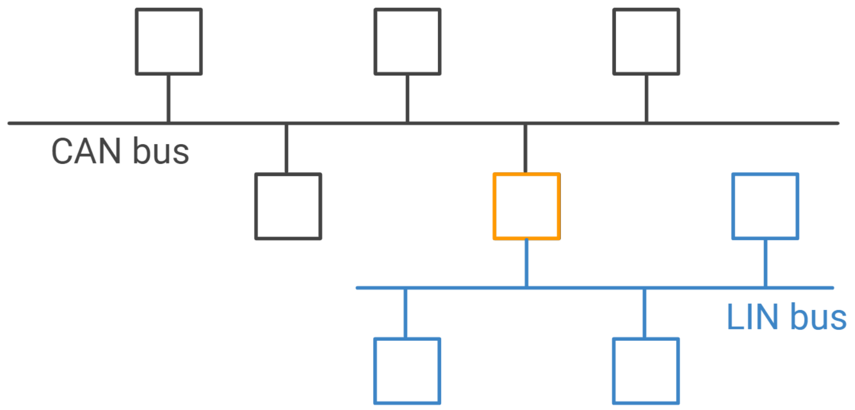


Figure A.1 – *Situation of the LIN bus with respect to the CAN bus [21]*

LIN buses are usually organized in clusters, each with a **master node** (the one that appears in orange in the figure A.1) that has a direct connection to the CAN bus.

In the figure A.2 we can see an example of a possible topology that buses could have inside a car. The example shows the implementation of the vehicle window control system, which is carried out with two LIN slaves belonging to two different LIN buses or clusters (with their respective masters) which in turn hang from a single CAN bus.

A.2.1.1 Example of LIN and CAN buses working inside a car

To illustrate the possible operation of this topology, the following situation is proposed:

"Usage scenario: In a car's right seat you can roll down the left seat window. To do so, you press a button to send a message via one LIN cluster to another LIN cluster via the CAN bus. This triggers the second LIN cluster to roll down the left seat window." [21]

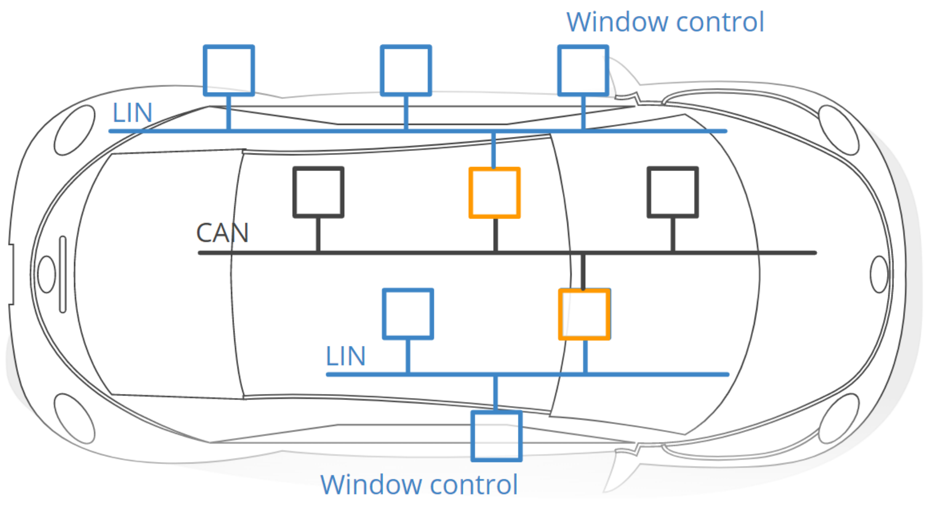


Figure A.2 – *Example of LIN and CAN buses topology inside a vehicle [21]*

A.2.2 Technical specifications of the LIN bus

In this section we will mention the technical characteristics and specifications of the LIN bus [26], [21]

- As we have said, the LIN bus is typically organized in clusters, each of which must have **1 master** and can have up to a maximum of **16 slaves**.
- It only need **one wire** to carry out the communication (in addition to the ground wire).
- The maximum length of the LIN cable to guarantee the absence of significant losses in communication is **40 meters**.
- It can reach transfer speeds of **up to 20 kb/s**.
- It's capable of working with different data field lengths (**2, 4 and 8 bytes**).
- It supports **error detection, checksum** [2] **and configuration**.
- It operates at **12 V**.
- Physical layer based on **ISO 9141 (K-line)**
- It supports **sleep mode and wake up** in order to reduce as much as possible power consumption.

A.2.3 LIN bus applications

Since LIN is not quite as reliable as CAN (among other things, because it doesn't have a bus arbitration) it's often used for applications that are **not time critical** or does not need **extreme fault tolerance**. The objective of LIN is to be easy to use and a more economic alternative to CAN. Some examples of applications where LIN can be used in a car are:

- **Headlights**: Control of different functions of the pilots. This is the application in which our work is focused.
- **Steering wheel**: Cruise control, wiper, climate control, radio, etc.
- **Comfort**: Sensors for temperature, sun roof, light or humidity.
- **Powertrain**: Sensors for position, speed and pressure
- **Engine**: Small motors and cooling fan motors
- **Air condition**: Motors, control panel, etc.
- **Door**: Side mirrors, windows, seat control, locks
- **Seats**: Position motors, pressure sensors

[2] Checksum: an operation at the bit level that is performed with certain bits transmitted in the frame whose result determines the correct reception of the same

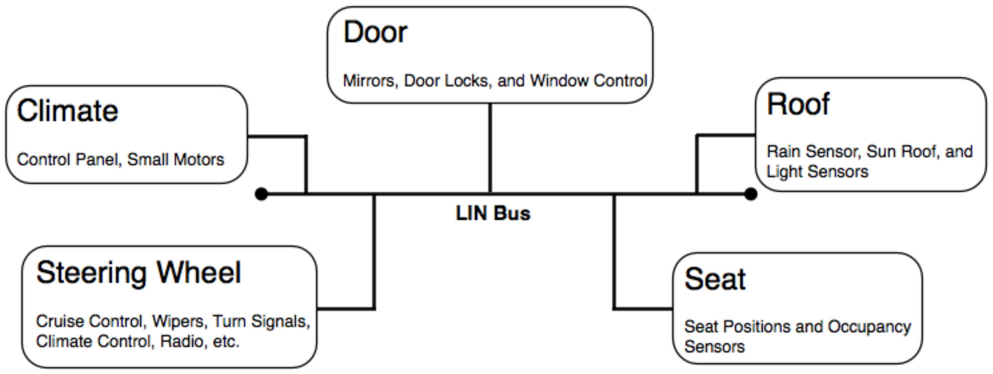

Door
Mirrors, Door Locks, and Window Control
Climate
Control Panel, Small Motors
Roof
Rain Sensor, Sun Roof, and Light Sensors
LIN Bus
Steering Wheel
Cruise Control, Wipers, Turn Signals, Climate Control, Radio, etc.
Seat
Seat Positions and Occupancy Sensors

Figure A.3 – *Diagram with some LIN bus functions inside a vehicle [27]*

A.3 LIN protocol

Once we know the characteristics and applications of the LIN bus at a physical level, we can delve into the LIN protocol at the frame structure level and the schedule followed for the correct operation of the communication.

A.3.1 LIN configuration file (LDF)

As we have already said, the architecture of a LIN bus consists of a master and one or more slaves (up to 16). One of the most important aspects to highlight in order to understand how the protocol works is the existence of two types of tasks, which are specified in a file called "Node Configuration File (NCF)". These files contains all the information about how each node should act when it receives a frame with a certain ID.

- **Slave task**: This task exists for the master node as well as for each of the slave nodes. It is responsible for responding to the header generated by the master node.

- **Master task**: This task only exists in the master node and is the one in charge of controlling all the communication on the LIN bus, being able to send headers so that the slaves to which they are addressed respond.

All the files belonging to the nodes are gathered in a single file that defines the behavior of all the communication in the LIN cluster called "**LIN Configuration File**" (LDF) [31]

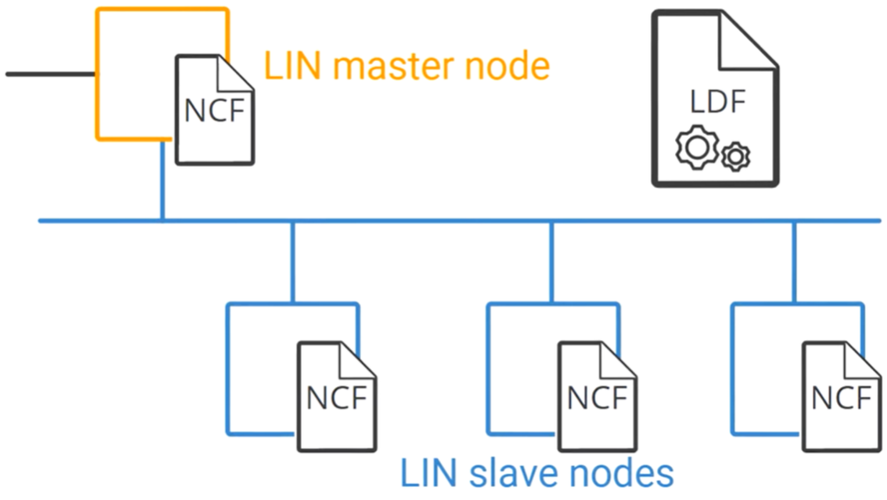


Figure A.4 – *NCF and LDF files in LIN cluster diagram [22]*

It should be noted that the importance of this section does not lie so much in the syntax of the LDF itself, but in really understanding how LIN communication works in detail, for which understanding this configuration file helps a lot.

LDF file mainly consists of 3 sections:

A.3.1.1 Header section

This section include the following information:

- Protocol version.
- Language version.
- Bus speed.
- Nodes
  - Master
    - Node name
    - Timer base
    - Jitter
  - Slave (Involved nodes)

Below is an example of what the header of an LDF could look like.

```

LIN_description_file;
LIN_protocol_version = "2.0";
LIN_language_version = "2.0";
LIN_speed = 10.4 Kbps;

Nodes {
    Master   :   Master ,  10ms,  0.1ms;
    Slave    :   Slave1 ;
}
```

Listado A.1 – *Example of header section o f an LDF file*

A.3.1.2 Signal definition section

In the LIN frame, the signals are a set of bits belonging to the data field that together represent a piece of data relevant for the "subscribers". Each signal is published by only one node, and it is always the same. None, one or more nodes can be "subscribed" to the signal, so they will attend to its value and act as ordered by their NCF. All signal in a LIN cluster are specified in its LDF.

This section contains the following information about signals:

- Signal name.
- Signal size.
- Initial value after POR
- Publisher: who sends and updates the signal (only one node)
- Subscriber(s): who reads the signal (multiples nodes possible)

Below is an example related to the windows control of a vehicle of what the signal definition section of the LDF could look like.

```

Signals{
    UP                :   1,  0.  Master ,  Slave1;
    DOWN              :   1,  0,  Master ,  Slave1;
    WindowBusInfo     :   8,  0,  Slave1 ,  Master;
    Window_error      :   8,  0,  Slave1 ,  Master;
    WindowLiftStatus:     8,  0,  Slave1 ,  Master;
}
```

Listado A.2 – *Example of signal definition section of an LDF file*

In this example, the first two signals are used to command the windows to go up or down. They are published by the master node and read by the slave node; they also have a length of 1 bit and their initial value is 0.

The other signals are related to the diagnosis of errors in the windows, so they are published by the slave node and read by the master node. They have a length of 1 byte (8 bits) and their initial value is 0.

A.3.1.3 Frame definition section

This section identify the name of all frames in the cluster as well as their properties. The definitions in this section create a frame identifier set (their symbolic name) and an associated frame ID set (the frame identifier). All frames in these sets shall be unique.

This section contains the following information:

- Frame name.
- Identifier: 0 to 64
- Publisher: node which publishes the frame header
- Response length: 8 bytes max. excluding checksum.
- Signals in response: defined in signal definition section A.3.1.2
- Offset: bit position where each signal starts in the response.

```

Frames{
    Window          :    5,  Master,  2 {
        UP          ,    0;
        DOWN        ,    1;
    }

    Response        :    53,  Slave1,  8 {
        WindowBusInfo,          16;
        WindowPanelInfo,        24;
        WindowError,            32;
        WindowLiftStatus,       40;
    }
}
```

Listado A.3 – *Example of frame definition section of an LDF file*

In this example, the behavior of two frames, one called "*Window*" and another one called "*Response*" is being defined. The frame identifier of "*Window*" is **ID = 0x05**, it is published by the **master** node and its data field is **2 bytes** length. The signals associated with the data field of this frame (**UP and DOWN** signals) are also specified, in addition to their start bit: **0 and 1 respectively** (LSB). Remember that a signal can only be published by one node.

The behavior of the *"Response"* frame is also defined, which will be generated by the slave node "**Slave1**" with a data field length of **8 bytes**. Finally, the signals associated with the data field of this frame are specified, as well as the start bit of each one (LSB).

Let us remember that in the signal definition section A.3.1.2, it has been specified which node is the receiver of each signal and, therefore, it is enabled to respond by writing in the data field of the frame. In this case, for the UP and DOWN signals present in the Window frame, the slave node "Slave1" will be in charge of generating the response. However, for the rest of the signals present in the "Response" frame, the master node will be in charge of responding to itself, as we must remember that it is also in charge of generating the frames.

The information that circulates through the LIN bus is organized in frames. Each frame has a header and a response to it.

A.3.2 LIN frame format

The information that circulates through the LIN bus is organized in frames. Each frame has a **header** and a **response** to it. In addition, there is a small time called "**response time**" that is left for the corresponding node to give time to respond.

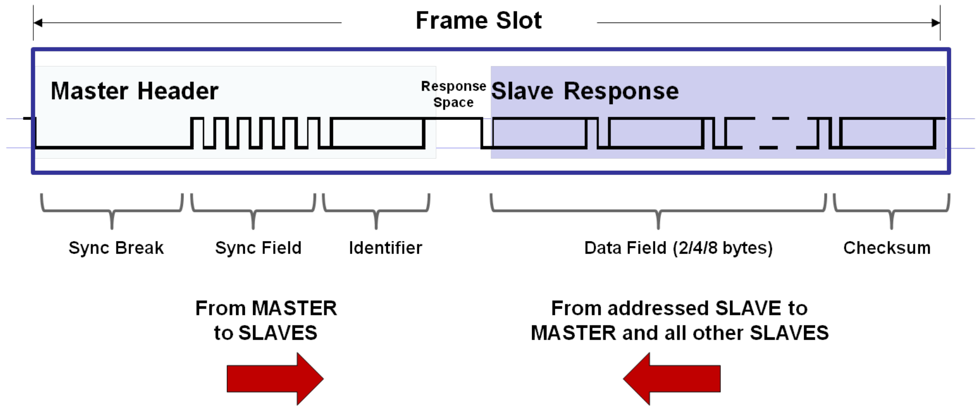


Figure A.5 – *LIN message frame [32]*

A.3.2.1 Header

It is always generated by the master and causes the corresponding slaves to generate a response. It has 3 fields.

A.3.2.1.1 Sync Break

It is used to indicate that a new frame begins. It must have a minimum duration of 13 times the bit time (Tb)

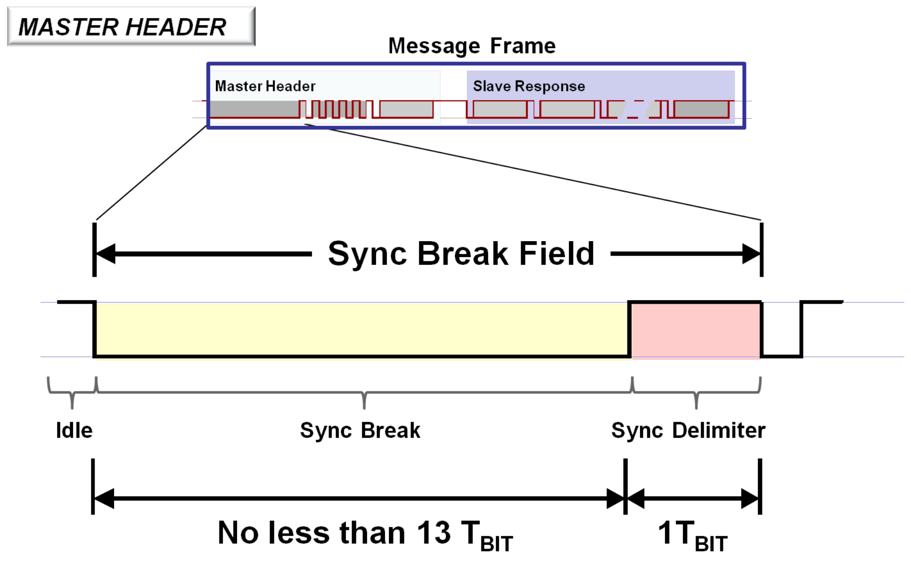


Figure A.6 – *Master header sync break field [32]*

A.3.2.1.2 Sync field

This field is used for the slave to know the time that elapses between two falling edges. In this way the slave knows the data transmission speed used by the master automatically (auto baud detection).

It occupies **1 byte** and always has the same value: **0x55 in hexadecimal**, or what is the same, **0b01010101** in binary.

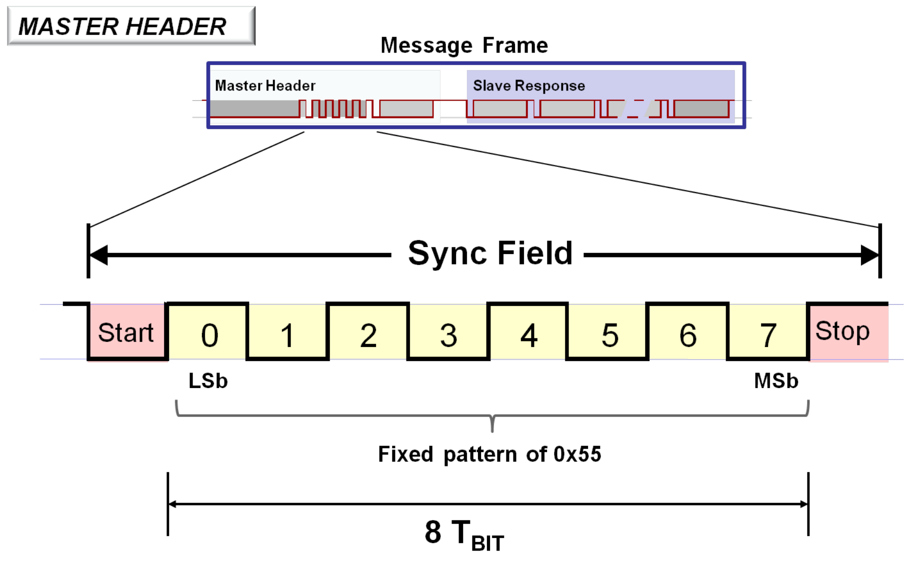


Figure A.7 – *Master header sync field*

A.3.2.1.3 Protected Identifier field (PID)

This field is divided into two parts or subfields.

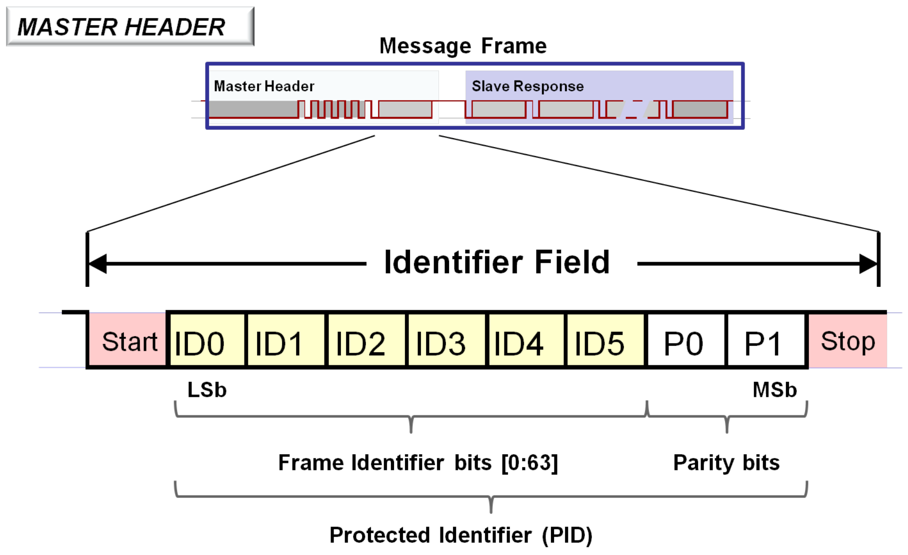


Figure A.8 – *Master header identifier field [32]*

Identifier field

It is made up of the 6 least significant bits (0 through 5) of the field. Contains information about the sender and receiver and the length in bytes expected in the response.

The table A.2 shows the expected length of the response data field based on what range the ID is in

ID Range (dec)	ID range (hex)	Data length
0-31	0x00-0x1F	2
32-47	0x20-0x2F	4
48-63	0x30-0x3F	8

Table A.2 – *Expected length of the data field based on the range of the ID [32]*

Parity field

It is made up of bits 6 and 7 and serves to check the integrity of the bits corresponding to the ID.

They are calculated with the following logical operations:

$P0 = ID0 \oplus ID1 \oplus ID2 \oplus ID4$

$P1 = ID1 \oplus ID3 \oplus ID4 \oplus ID5$

A.3.2.2 Response

In most cases the signals present in the response field are generated by the same slave to avoid collisions, since, as we have said in A.3.2. The LIN protocol does not have any arbitration.

A.3.2.2.1 Data field

Based on the transmitted identifier, a Slave will recognize that it's been addressed, and reply with a response in the data (signal) field. Several signals can be packed into one frame. Data bytes are transmitted LSB first.

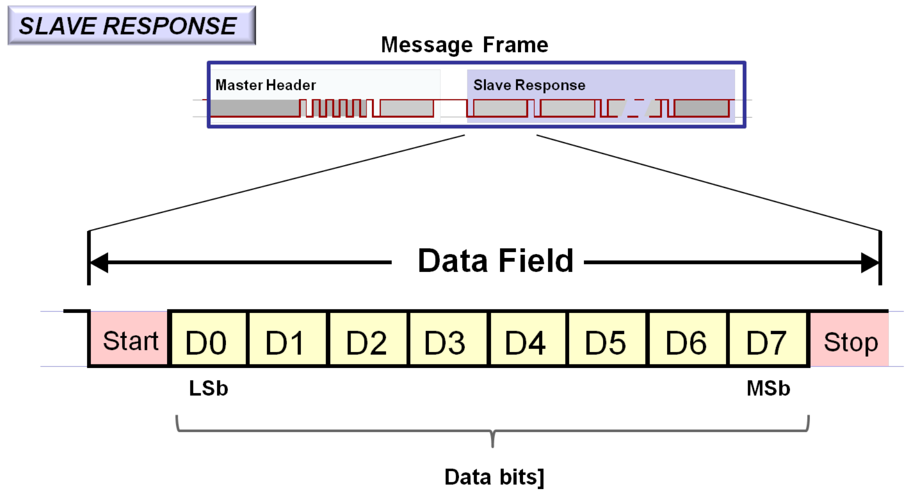


Figure A.9 – *Slave response data field [32]*

The response can contain from 1 to 8 data fields

A.3.2.2.2 Checksum field

The data response of the Slaves is checked by means of a checksum.

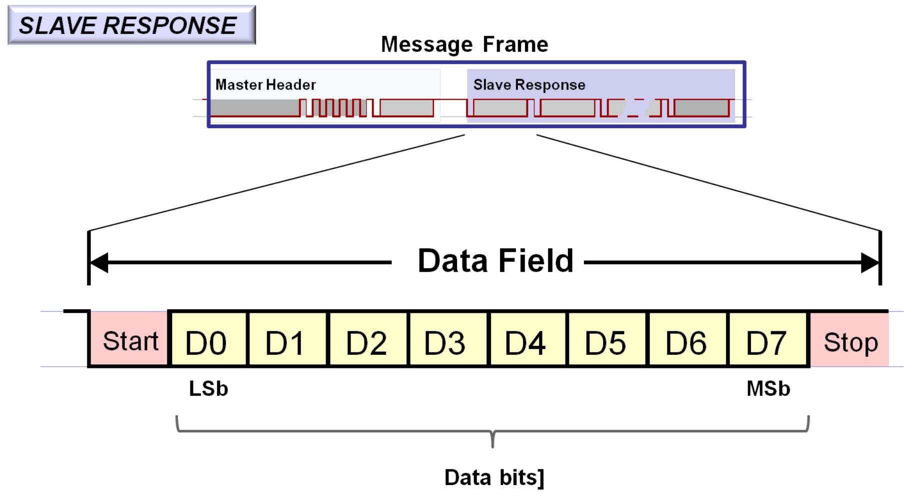
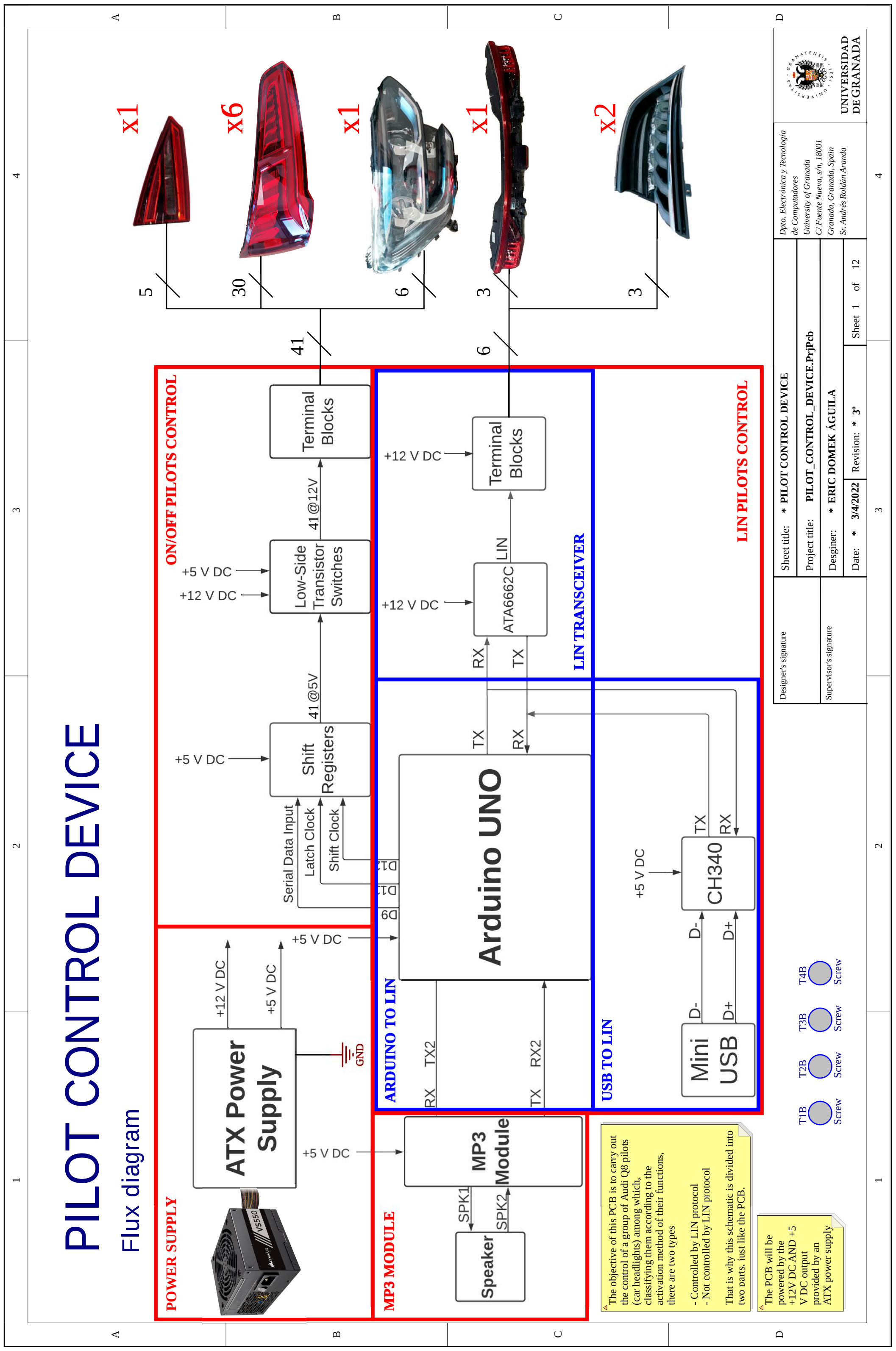


Figure A.10 – *Slave response checksum field [32]*

Appendix B

Altium files

B.1 Electronic Schematics



The objective of this PCB is to carry out the control of a group of Audi Q8 pilots (car headlights) among which, classifying them according to the activation method of their functions, there are two types

- Controlled by LIN protocol
- Not controlled by LIN protocol

That is why this schematic is divided into two parts, just like the PCB.

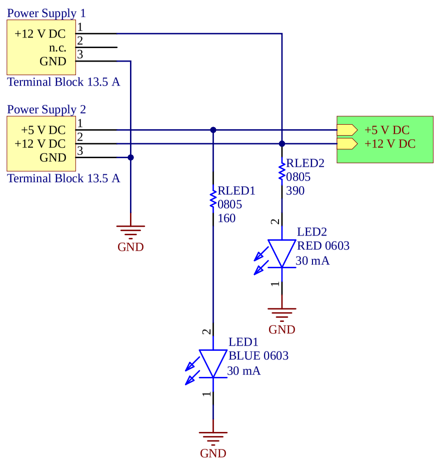
The PCB will be powered by the +12V DC AND +5 V DC output provided by an ATX power supply

Designer's signature	Sheet title: * PILOT CONTROL DEVICE		Dpto. Electrónica y Tecnología de Computadores University of Granada C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain Sr. Andrés Roldán Aranda
	Project title: PILOT_CONTROL_DEVICE.PrjPcb		UNIVERSIDAD DE GRANADA
Supervisor's signature	Desginer: * ERIC DOMEK ÁGUILA		
	Date: * 3/4/2022	Revision: * 3º	Sheet 1 of 12

POWER SUPPLY

The +5 V DC required to power the ATA6662C, the CH340 and the Arduino comes from this NCV8518B linear regulator that provides +5 V DC output with a 0 to +45 V DC input.

As specified in the characteristics sheet of the NCV8518B, the ENABLE pin makes the IC work when EN = 1. Since this function is not going to be used, we put it high through a resistor of the recommended value.



Power budget

PILOTOS TRASEROS	Trunk (largo)	Fender	Trunk Lamp	Trunk LHD
Freno (A)	0.34	0.4	0.28	0.22
Indicador (A)	0.3	0.25	0.51	0.29
Posición (A)	0.54	0.26	0.37	0.07
Marcha atrás (A)	0.79	-	-	0.2
Número	1	2	6	1
Consumo máx. (A)	1.97	1.82	6.96	0.78

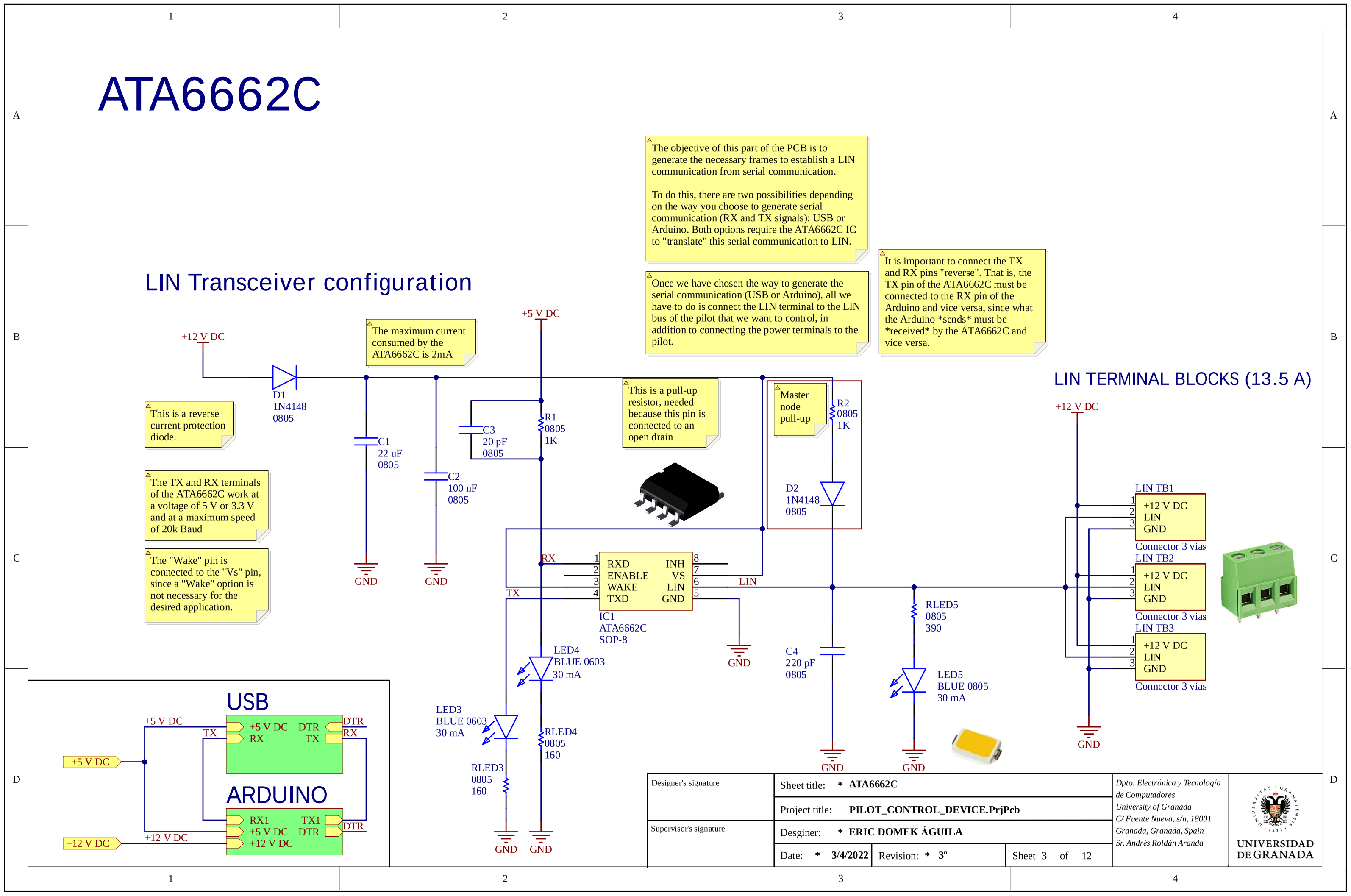
FARO DELANTERO	
DRL (A)	1.36
LB (A)	3.9
HB (A)	3.9
TI (A)	-
Número	1
Consumo máx. (A)	9.16

OTROS COMPONENTES	LEDs	ATA6662C	CH340	DFPLAYER	74HC595
Típico (A)	0.03	0.0012	0.012	0.02	0.000001
Máximo (A)	0.03	0.002	0.03	0.02	0.000001
Mínimo (A)	0.015	0.00002	0.012	0.02	0.0000001
Número	32	1	1	1	4
Consumo máx. (A)	0.96	0.002	0.03	0.02	0.000004
Consumo mín. (A)	0.48	0.00002	0.012	0.02	0.0000004
Consumo típ. (A)	0.96	0.0012	0.012	0.02	0.000004

Consumo tot. LIN (A)	3.79
Consumo tot. ON/OFF (A)	16.9
Consumo tot. máx. (A)	21.702004

Designer's signature	Sheet title: * POWER SUPPLY	Dpto. Electrónica y Tecnología de Computadores
	Project title: PILOT_CONTROL_DEVICE.PrjPcb	University of Granada
Supervisor's signature	Desginer: * ERIC DOMEK ÁGUILA	C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain
	Date: * 3/4/2022 Revision: * 3º Sheet 2 of 12	Sr. Andrés Roldán Aranda

UNIVERSIDAD DE GRANADA



This is a reverse current protection diode.

The TX and RX terminals of the ATA6662C work at a voltage of 5 V or 3.3 V and at a maximum speed of 20k Baud

The "Wake" pin is connected to the "Vs" pin, since a "Wake" option is not necessary for the desired application.

The maximum current consumed by the ATA6662C is 2mA

This is a pull-up resistor, needed because this pin is connected to an open drain

Master node pull-up

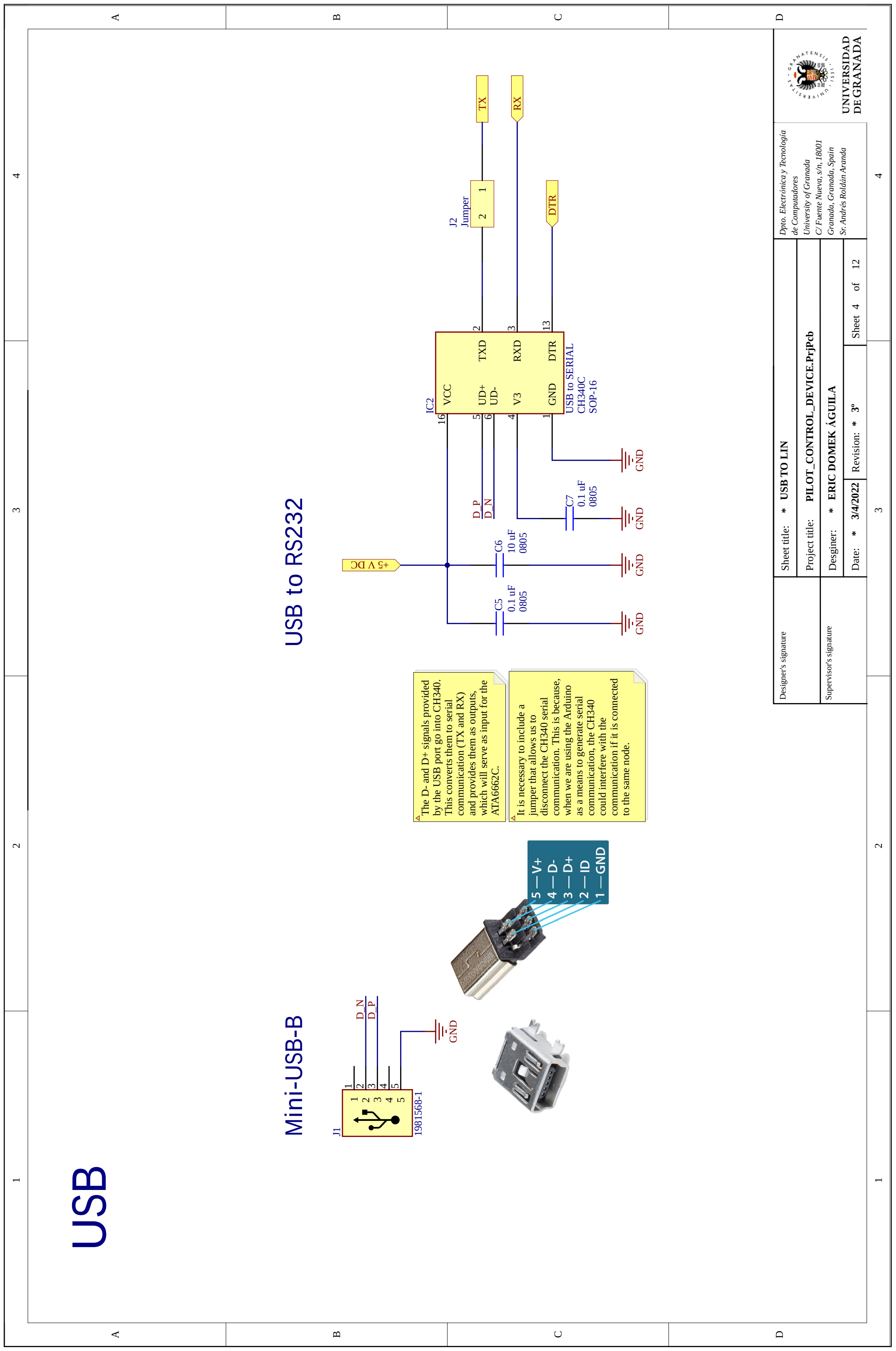
The objective of this part of the PCB is to generate the necessary frames to establish a LIN communication from serial communication.

To do this, there are two possibilities depending on the way you choose to generate serial communication (RX and TX signals): USB or Arduino. Both options require the ATA6662C IC to "translate" this serial communication to LIN.

Once we have chosen the way to generate the serial communication (USB or Arduino), all we have to do is connect the LIN terminal to the LIN bus of the pilot that we want to control, in addition to connecting the power terminals to the pilot.

It is important to connect the TX and RX pins "reverse". That is, the TX pin of the ATA6662C must be connected to the RX pin of the Arduino and vice versa, since what the Arduino *sends* must be *received* by the ATA6662C and vice versa.

Sheet title:	* ATA6662C
Project title:	PILOT_CONTROL_DEVICE.PrjPcb
Desginer:	* ERIC DOMEK ÁGUILA
Date: * 3/4/2022	Revision: * 3º / Sheet 3 of 12



The D- and D+ signals provided by the USB port go into CH340. This converts them to serial communication (TX and RX) and provides them as outputs, which will serve as input for the ATA6662C.

It is necessary to include a jumper that allows us to disconnect the CH340 serial communication. This is because, when we are using the Arduino as a means to generate serial communication, the CH340 could interfere with the communication if it is connected to the same node.

Designer's signature

Supervisor's signature

Sheet title: * USB TO LIN

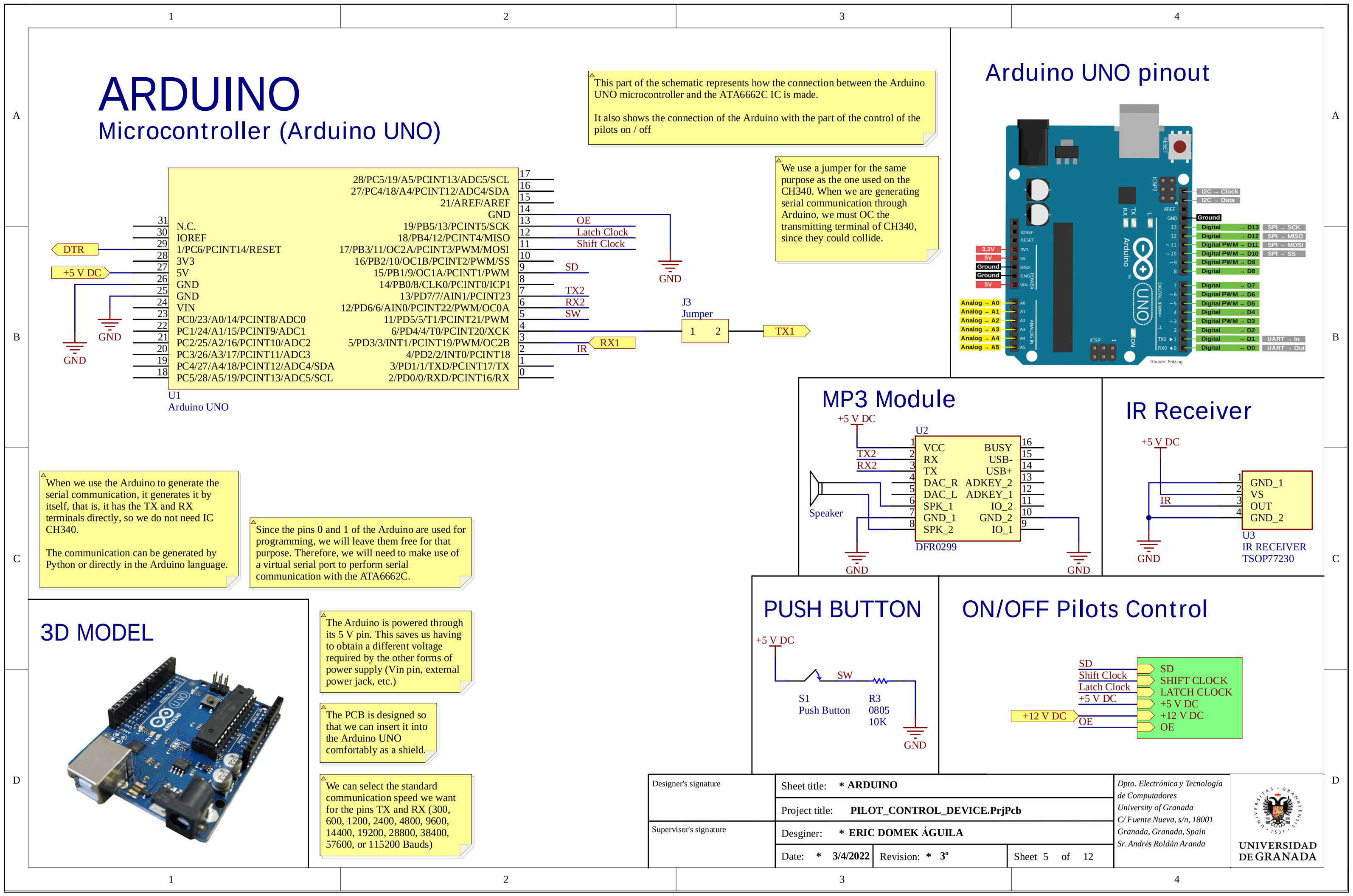
Project title: PILOT_CONTROL_DEVICE.PrjPcb

Desginer: * ERIC DOMEK ÁGUILA

Date: * 3/4/2022 Revision: * 3ª Sheet 4 of 12

Dpto. Electrónica y Tecnología de Computadores
University of Granada
C/ Fuente Nueva, s/n, 18001
Granada, Granada, Spain
Sr. Andrés Roldán Aranda

UNIVERSIDAD DE GRANADA



This part of the schematic represents how the connection between the Arduino UNO microcontroller and the ATA6662C IC is made.

It also shows the connection of the Arduino with the part of the control of the pilots on / off

We use a jumper for the same purpose as the one used on the CH340. When we are generating serial communication through Arduino, we must OC the transmitting terminal of CH340, since they could collide.

When we use the Arduino to generate the serial communication, it generates it by itself, that is, it has the TX and RX terminals directly, so we do not need IC CH340.

The communication can be generated by Python or directly in the Arduino language.

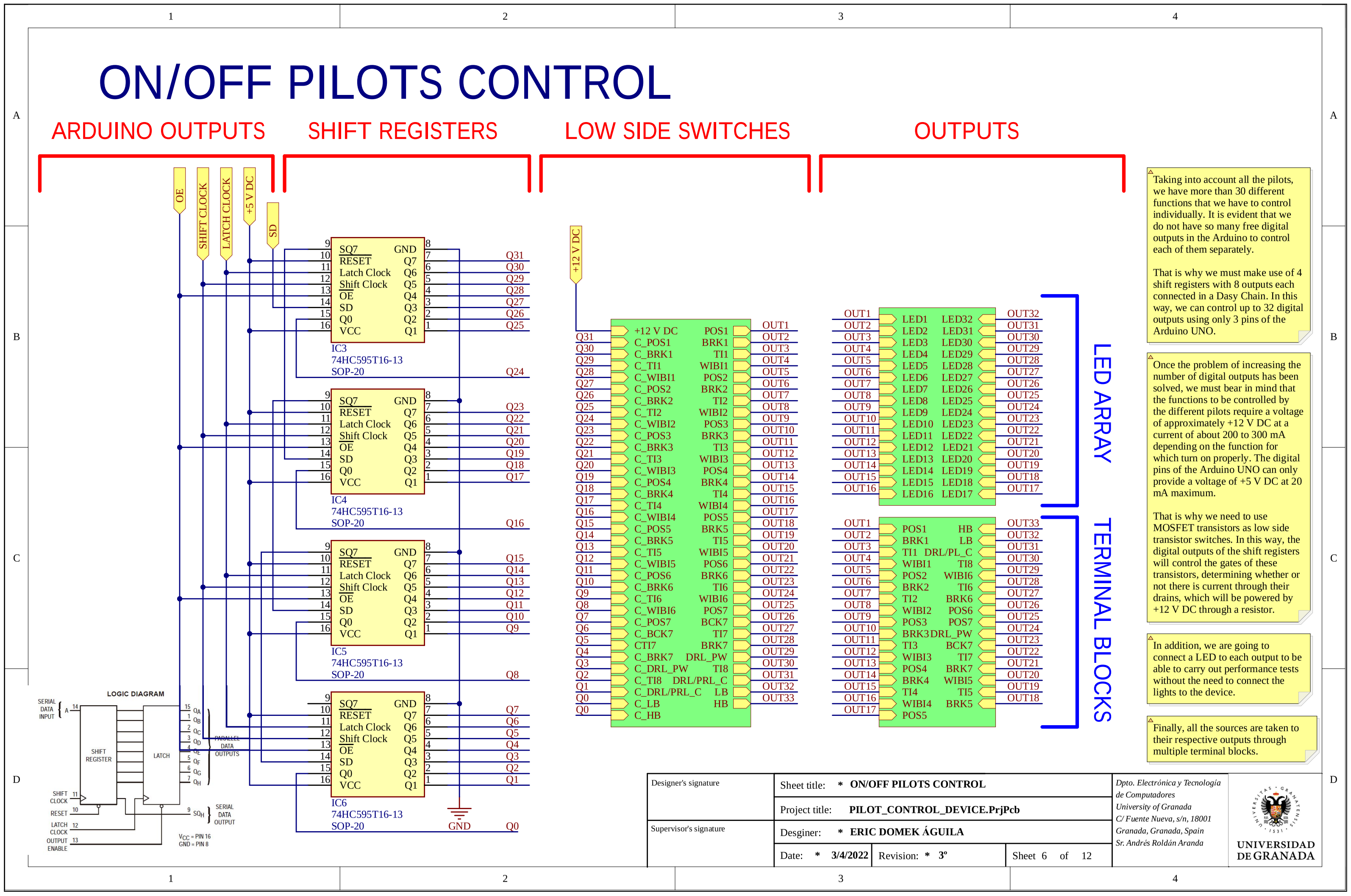
Since the pins 0 and 1 of the Arduino are used for programming, we will leave them free for that purpose. Therefore, we will need to make use of a virtual serial port to perform serial communication with the ATA6662C.

The Arduino is powered through its 5 V pin. This saves us having to obtain a different voltage required by the other forms of power supply (Vin pin, external power jack, etc.)

The PCB is designed so that we can insert it into the Arduino UNO comfortably as a shield.

We can select the standard communication speed we want for the pins TX and RX (300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200 Bauds)

Designer's signature	Sheet title: * ARDUINO		Dpto. Electrónica y Tecnología de Computadores University of Granada C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain Sr. Andrés Roldán Aranda
	Project title: PILOT_CONTROL_DEVICE.PrjPcb		
Supervisor's signature	Desginer: * ERIC DOMEK ÁGUILA		UNIVERSIDAD DE GRANADA
	Date: * 3/4/2022 Revision: * 3º	Sheet 5 of 12	



Taking into account all the pilots, we have more than 30 different functions that we have to control individually. It is evident that we do not have so many free digital outputs in the Arduino to control each of them separately.

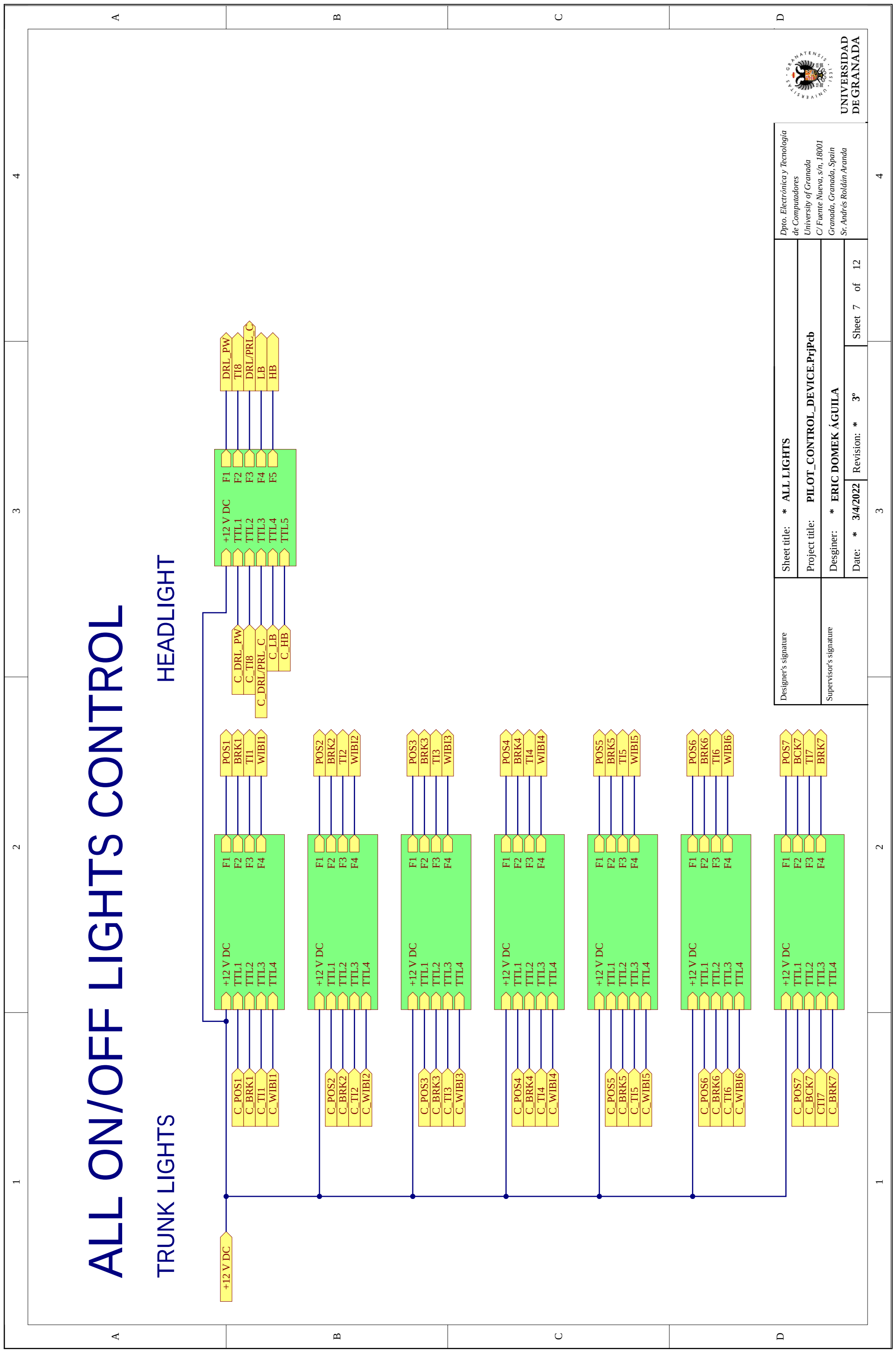
That is why we must make use of 4 shift registers with 8 outputs each connected in a Dasy Chain. In this way, we can control up to 32 digital outputs using only 3 pins of the Arduino UNO.

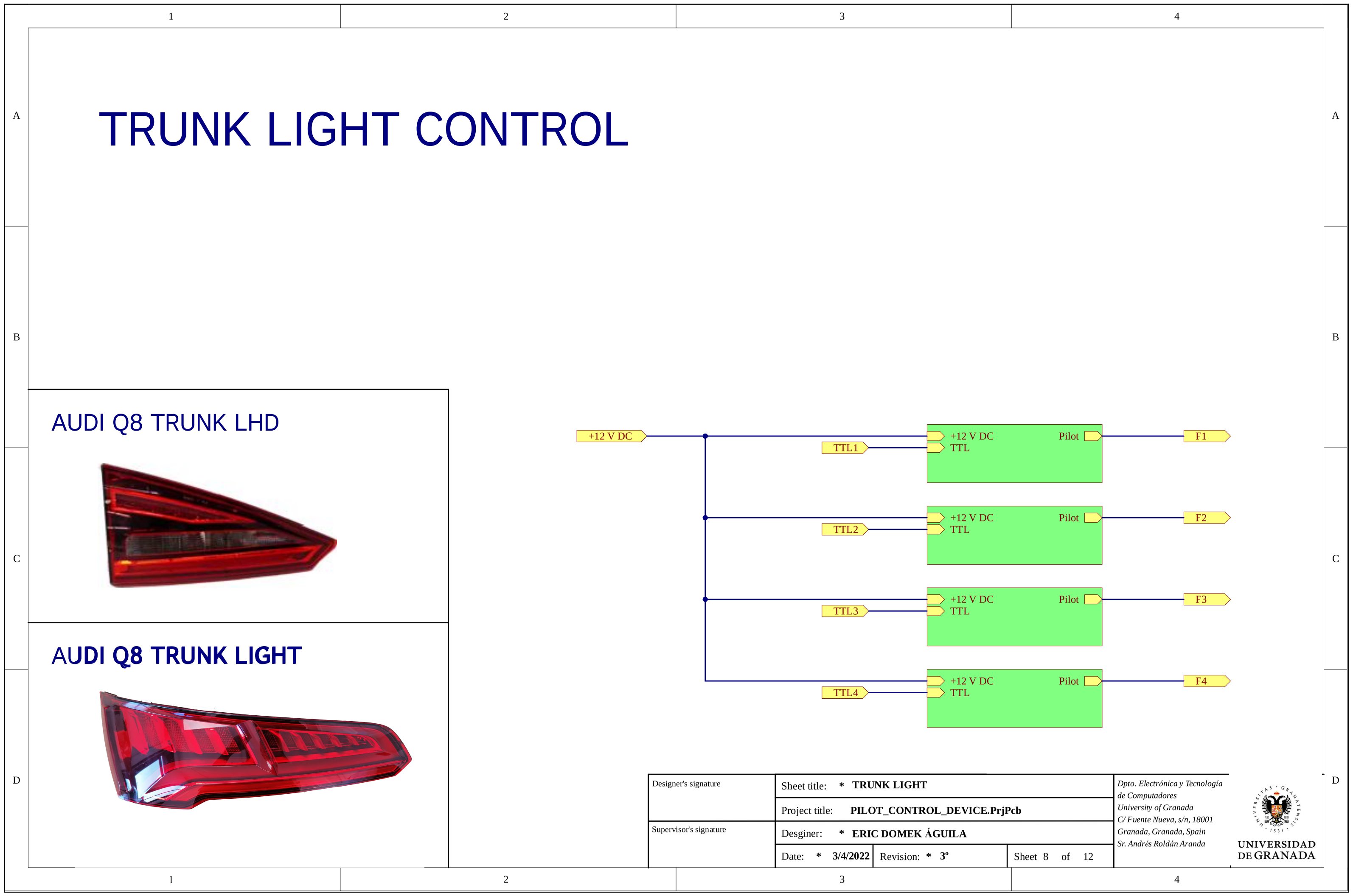
Once the problem of increasing the number of digital outputs has been solved, we must bear in mind that the functions to be controlled by the different pilots require a voltage of approximately +12 V DC at a current of about 200 to 300 mA depending on the function for which turn on properly. The digital pins of the Arduino UNO can only provide a voltage of +5 V DC at 20 mA maximum.

That is why we need to use MOSFET transistors as low side transistor switches. In this way, the digital outputs of the shift registers will control the gates of these transistors, determining whether or not there is current through their drains, which will be powered by +12 V DC through a resistor.

In addition, we are going to connect a LED to each output to be able to carry out performance tests without the need to connect the lights to the device.

Finally, all the sources are taken to their respective outputs through multiple terminal blocks.


ALL ON/OFF LIGHTS CONTROL
TRUNK LIGHTS
HEADLIGHT
+12 V DC
TTL1
TTL2
TTL3
TTL4
TTL5
F1
F2
F3
F4
F5
C_DRL_PW
C_TI8
C_DRL/PRL_C
C_LB
C_HB
DRL_PW
TI8
DRL/PRL_C
LB
HB
C_POS1
C_BRK1
C_TI1
C_WIBI1
POS1
BRK1
TI1
WIBI1
C_POS2
C_BRK2
C_TI2
C_WIBI2
POS2
BRK2
TI2
WIBI2
C_POS3
C_BRK3
C_TI3
C_WIBI3
POS3
BRK3
TI3
WIBI3
C_POS4
C_BRK4
C_TI4
C_WIBI4
POS4
BRK4
TI4
WIBI4
C_POS5
C_BRK5
C_TI5
C_WIBI5
POS5
BRK5
TI5
WIBI5
C_POS6
C_BRK6
C_TI6
C_WIBI6
POS6
BRK6
TI6
WIBI6
C_POS7
C_BCK7
CTI7
C_BRK7
POS7
BCK7
TI7
BRK7
Designer's signature
Supervisor's signature
Sheet title: * ALL LIGHTS
Project title: PILOT_CONTROL_DEVICE.PrjPcb
Desginer: * ERIC DOMEK ÁGUILA
Date: * 3/4/2022
Revision: * 3ª
Sheet 7 of 12
Dpto. Electrónica y Tecnología de Computadores
University of Granada
C/ Fuente Nueva, s/n, 18001
Granada, Granada, Spain
Sr. Andrés Roldán Aranda
UNIVERSIDAD DE GRANADA
A
B
C
D
1
2
3
4



Designer's signature	Sheet title: * **TRUNK LIGHT**	*Dpto. Electrónica y Tecnología de Computadores*
	Project title: **PILOT_CONTROL_DEVICE.PrjPcb**	*University of Granada*
Supervisor's signature	Desginer: * **ERIC DOMEK ÁGUILA**	*C/ Fuente Nueva, s/n, 18001* *Granada, Granada, Spain*
	Date: * **3/4/2022** Revision: * **3º** Sheet 8 of 12	*Sr. Andrés Roldán Aranda*

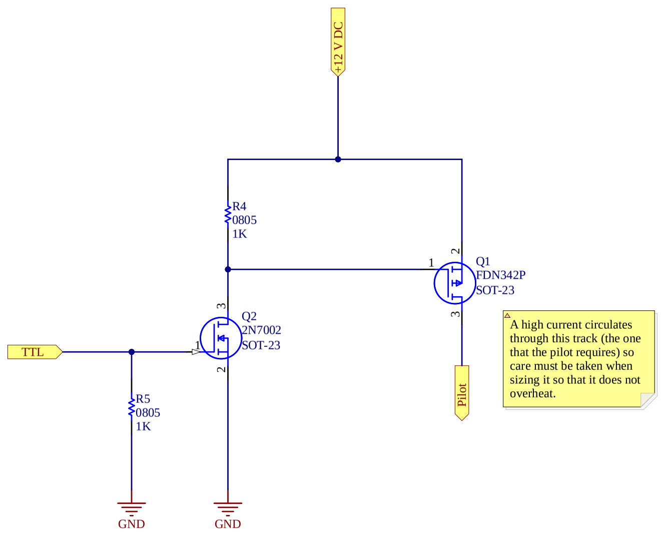
UNIVERSIDAD DE GRANADA

LOW SIDE SWITCH

This circuit makes it possible to control the presence or absence of a +12 V DC voltage from a TTL control signal (0 or 5 V). The transistor models used allow a suitable current for the desired application

When performing a reset in our design, it is possible that the TTL signal (0-5V) remains unconnected for a short period of time, leaving the gate of the mosfet transistor "on the air".

The purpose of the pull-down resistor R7 between the gate and the source of the nmos transistor is to close the circuit when this happens, thus being able to turn off the pilot.



A high current circulates through this track (the one that the pilot requires) so care must be taken when sizing it so that it does not overheat.

2N7002

● MAXIMUM RATING (Ta = 25℃)

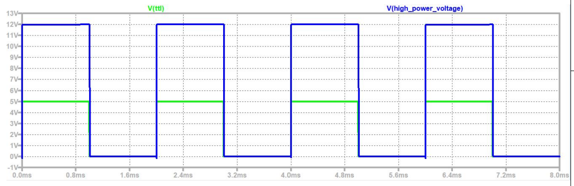
Rating	Symbol	Value	Unit
Drain–Source Voltage	V_{DSS}	60	Vdc
Drain–Gate Voltage (R_{GS} = 1.0 MΩ)	V_{DGR}	60	Vdc
Drain Current – Continuous T_C = 25°C (Note 1.) T_C = 100°C (Note 1.) – Pulsed (Note 2.)	I_D I_D I_{DM}	± 115 ± 75 ± 800	mAdc
Gate–Source Voltage – Continuous – Non–repetitive (t_p ≤ 50 μs)	 V_{GS} V_{GSM}	 ± 20 ± 40	 Vdc Vpk

FDN342P

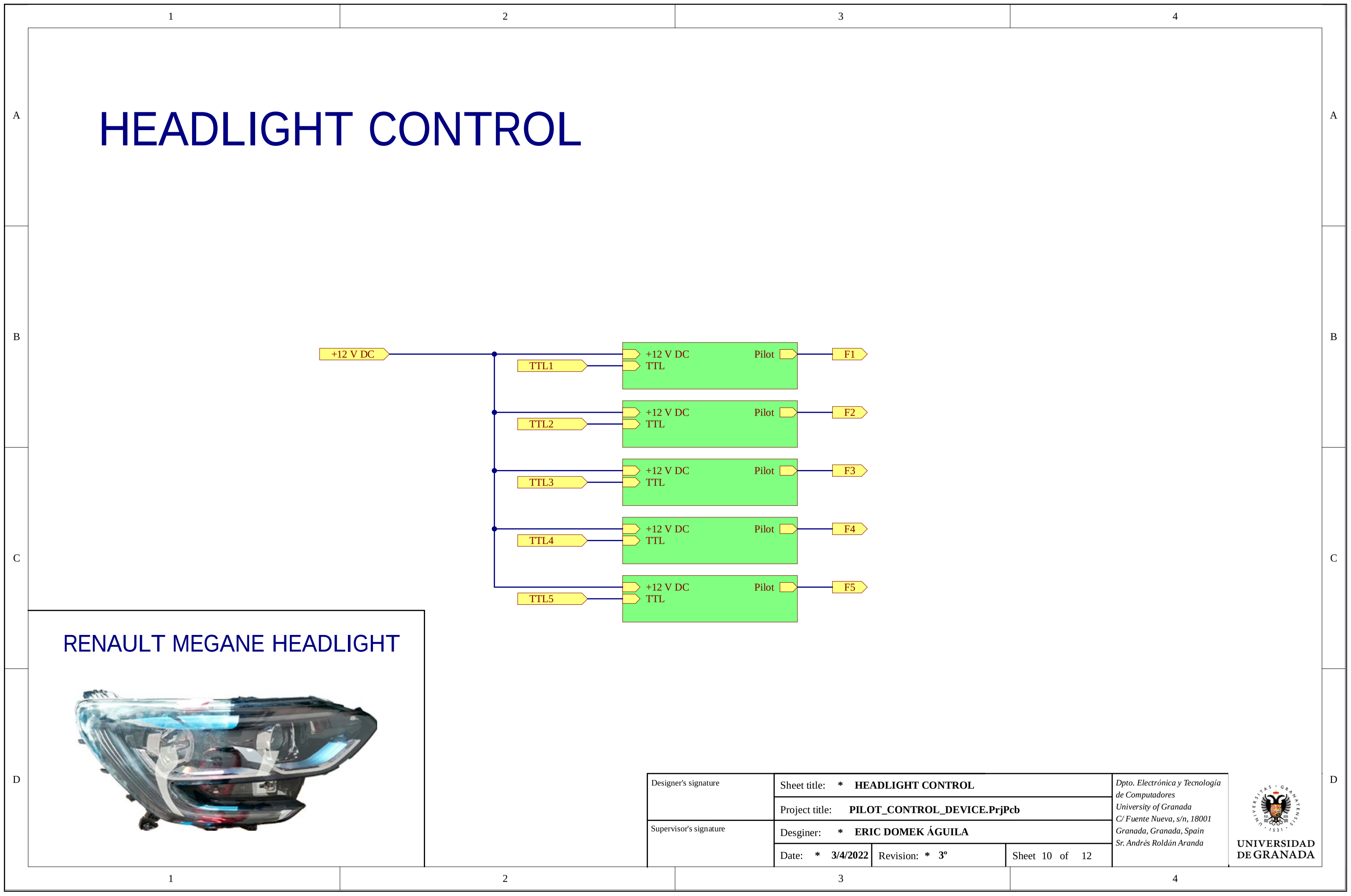
Absolute Maximum Ratings T_A = 25°C unless otherwise noted

Symbol	Parameter		Ratings	Units
V_{DSS}	Drain-Source Voltage		-20	V
V_{GSS}	Gate-Source Voltage		±12	V
I_D	Drain Current - Continuous	(Note 1a)	-2	A
	- Pulsed		-10	
P_D	Power Dissipation for Single Operation	(Note 1a)	0.5	W
		(Note 1b)	0.46	
T_J, T_{stg}	Operating and Storage Junction Temperature Range		-55 to +150	°C

Electrical simulation (Vpilot)

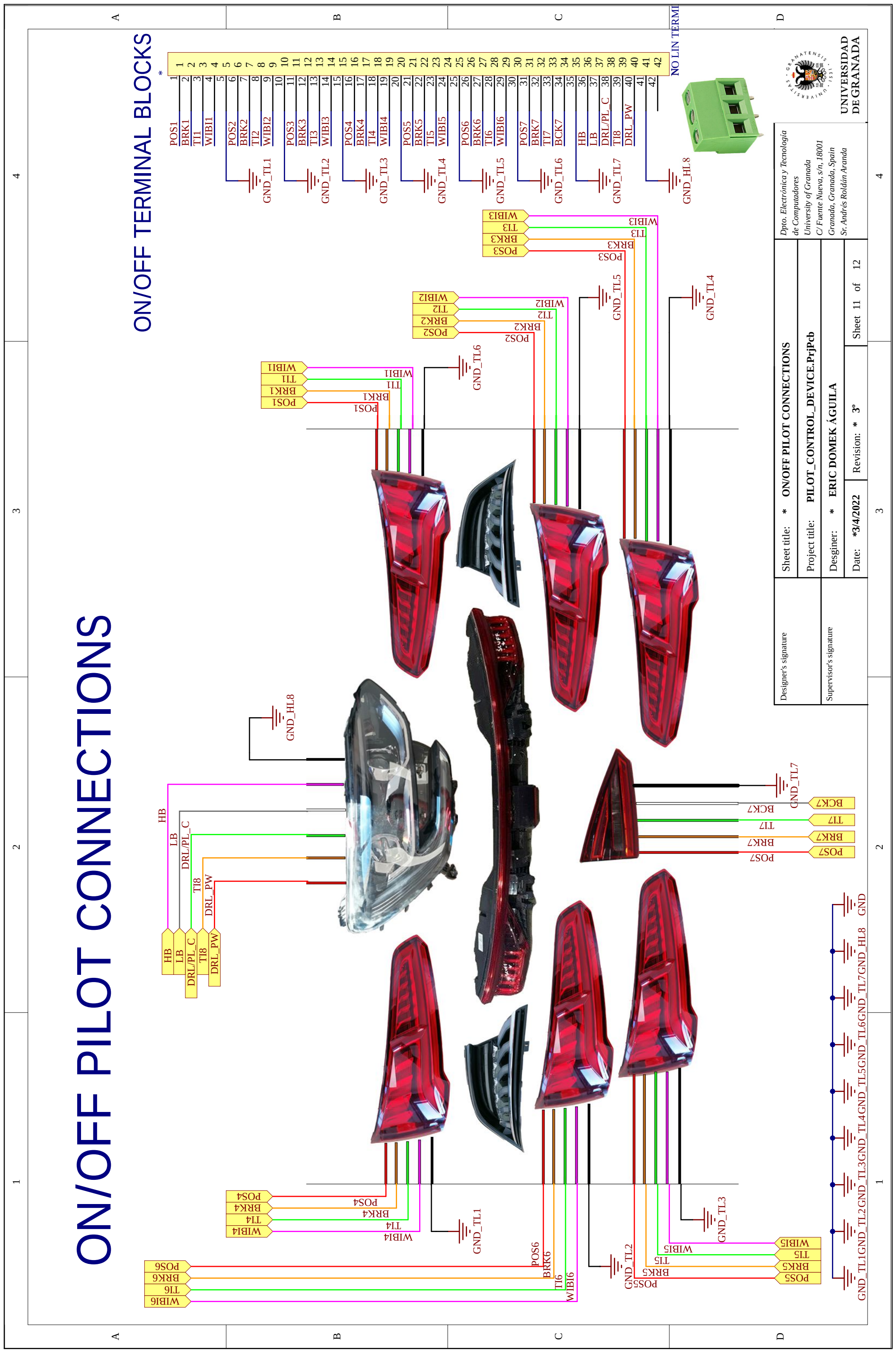


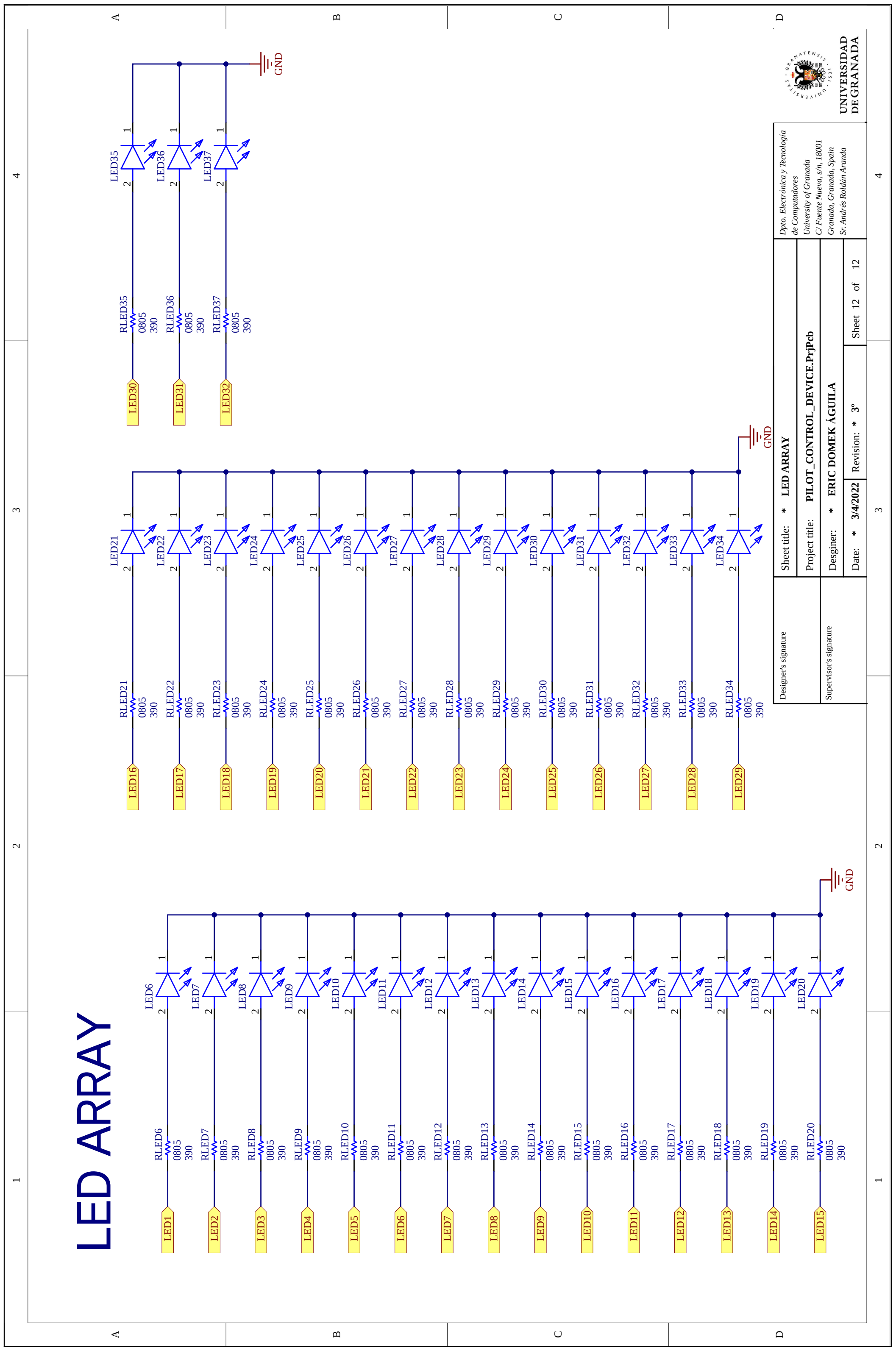
Designer's signature	Sheet title: * LOW SIDE SWITCHES		Dpto. Electrónica y Tecnología de Computadores University of Granada C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain Sr. Andrés Roldán Aranda
	Project title: PILOT_CONTROL_DEVICE.PrjPcb		
Supervisor's signature	Desginer: * ERIC DOMEK ÁGUILA		
	Date: * 22/03/2022 Revision: * 2°	Sheet 9 of 12	UNIVERSIDAD DE GRANADA



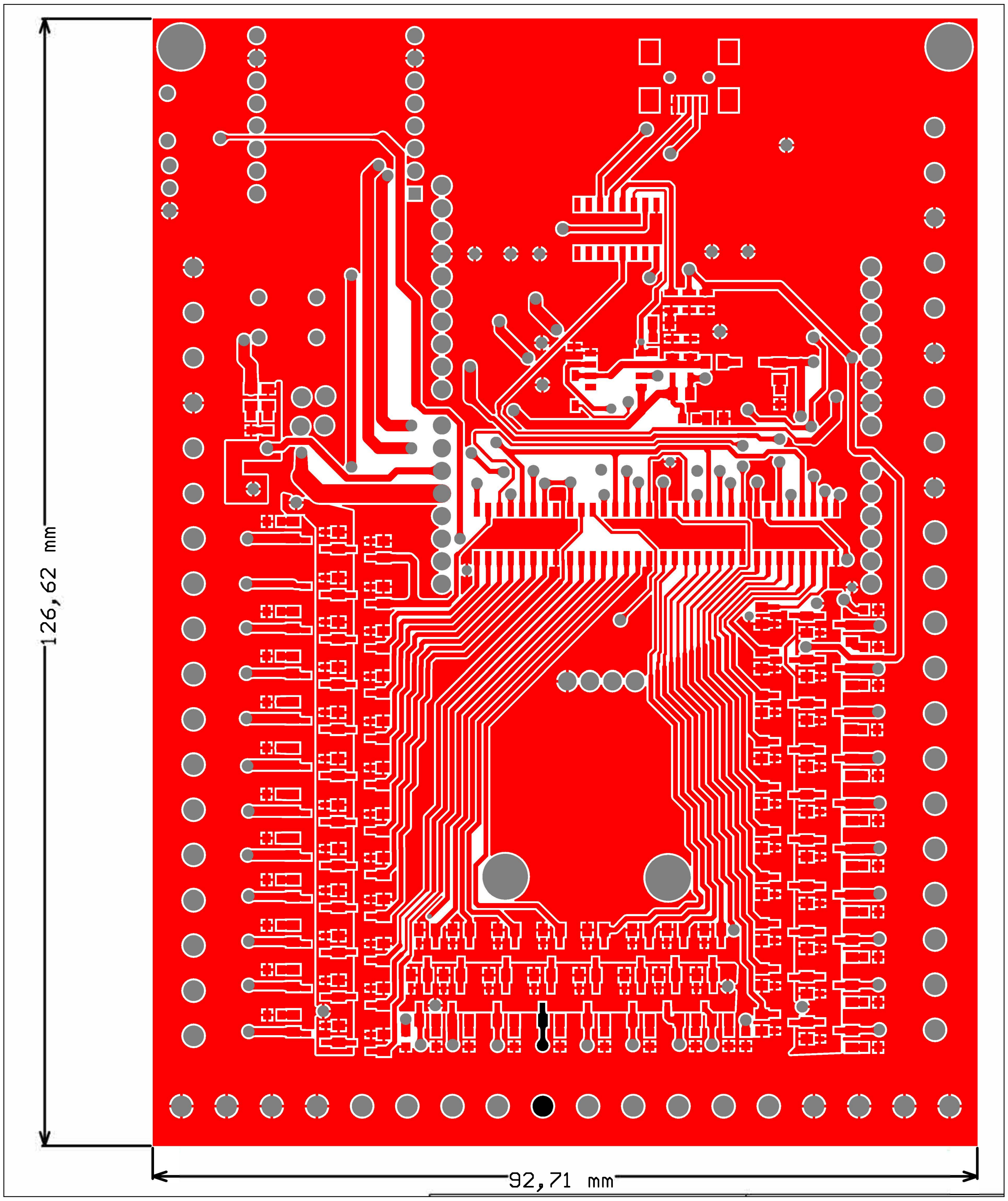
Designer's signature	Sheet title: * HEADLIGHT CONTROL	*Dpto. Electrónica y Tecnología de Computadores* *University of Granada* *C/ Fuente Nueva, s/n, 18001* *Granada, Granada, Spain* *Sr. Andrés Roldán Aranda*
	Project title: PILOT_CONTROL_DEVICE.PrjPcb	
Supervisor's signature	Desginer: * ERIC DOMEK ÁGUILA	
	Date: * 3/4/2022 Revision: * 3º Sheet 10 of 12	

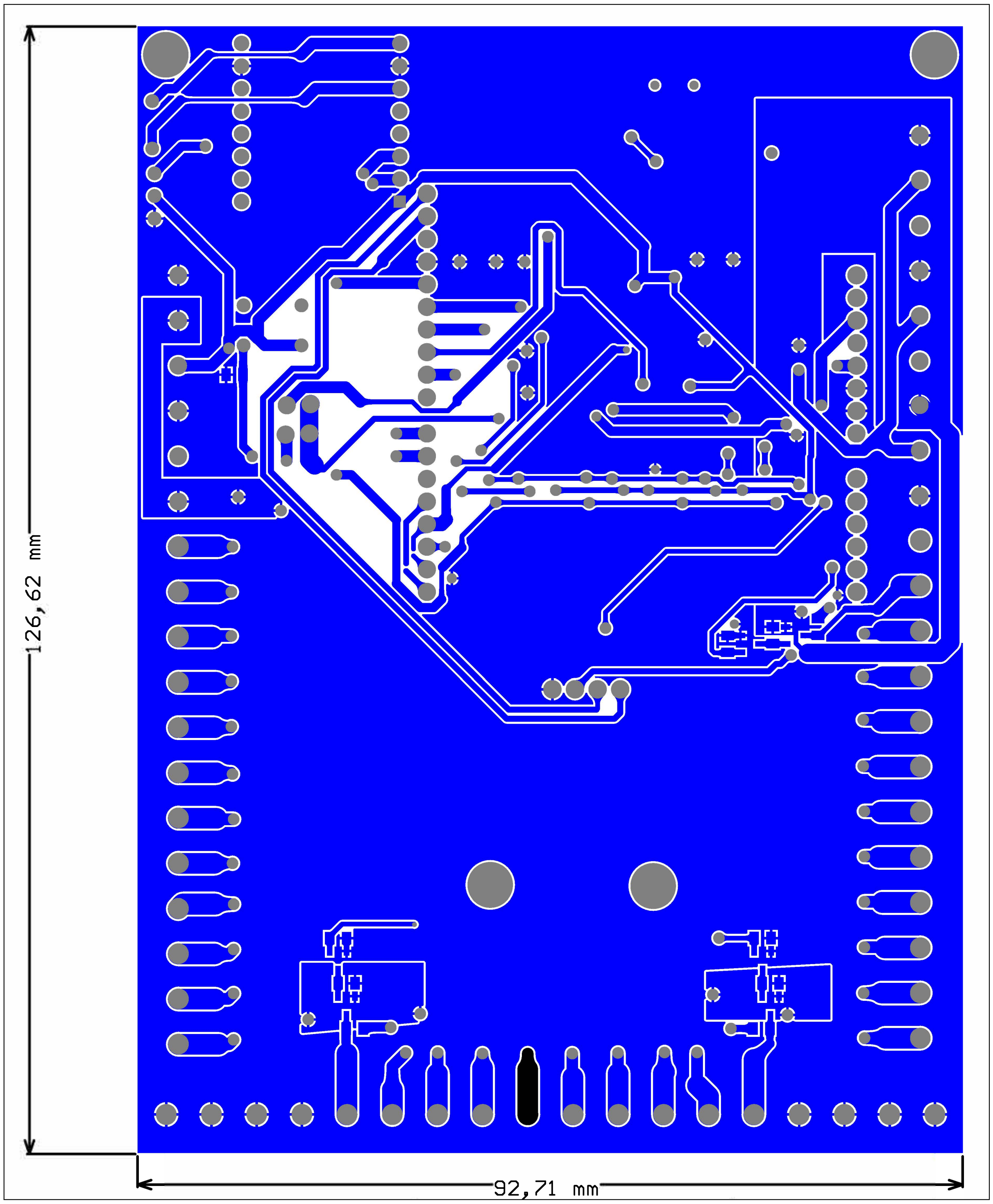
UNIVERSIDAD DE GRANADA


ON/OFF PILOT CONNECTIONS
ON/OFF TERMINAL BLOCKS
POS1 BRK1 TI1 WIBI1
POS2 BRK2 TI2 WIBI2
POS3 BRK3 TI3 WIBI3
POS4 BRK4 TI4 WIBI4
POS5 BRK5 TI5 WIBI5
POS6 BRK6 TI6 WIBI6
POS7 BRK7 TI7 BCK7
HB LB DRL/PL_C TI8 DRL_PW
GND_TL1 GND_TL2 GND_TL3 GND_TL4 GND_TL5 GND_TL6 GND_TL7 GND_HL8 GND
NO LIN TERMI
Designer's signature
Supervisor's signature
Sheet title: * ON/OFF PILOT CONNECTIONS
Project title: PILOT_CONTROL_DEVICE.PrjPcb
Desginer: * ERIC DOMEK ÁGUILA
Date: *3/4/2022
Revision: * 3*
Sheet 11 of 12
Dpto. Electrónica y Tecnología de Computadores
University of Granada
C/ Fuente Nueva, s/n, 18001
Granada, Granada, Spain
Sr. Andrés Roldán Aranda
UNIVERSIDAD DE GRANADA


LED ARRAY
LED1
LED2
LED3
LED4
LED5
LED6
LED7
LED8
LED9
LED10
LED11
LED12
LED13
LED14
LED15
RLED6
RLED7
RLED8
RLED9
RLED10
RLED11
RLED12
RLED13
RLED14
RLED15
RLED16
RLED17
RLED18
RLED19
RLED20
0805
390
LED16
LED17
LED18
LED19
LED20
LED21
LED22
LED23
LED24
LED25
LED26
LED27
LED28
LED29
RLED21
RLED22
RLED23
RLED24
RLED25
RLED26
RLED27
RLED28
RLED29
RLED30
RLED31
RLED32
RLED33
RLED34
LED30
LED31
LED32
LED33
LED34
LED35
LED36
LED37
RLED35
RLED36
RLED37
GND
Designer's signature
Supervisor's signature
Sheet title: * LED ARRAY
Project title: PILOT_CONTROL_DEVICE.PrjPcb
Desginer: * ERIC DOMEK ÁGUILA
Date: * 3/4/2022
Revision: * 3ª
Sheet 12 of 12
Dpto. Electrónica y Tecnología de Computadores
University of Granada
C/ Fuente Nueva, s/n, 18001
Granada, Granada, Spain
Sr. Andrés Roldán Aranda
UNIVERSIDAD DE GRANADA
A
B
C
D
1
2
3
4

B.2 PCB 2D


126,62 mm
92,71 mm


126,62 mm
92,71 mm

Appendix C

Drivers installation guide

C.1 Exxotest USB-MUXDIAG-II

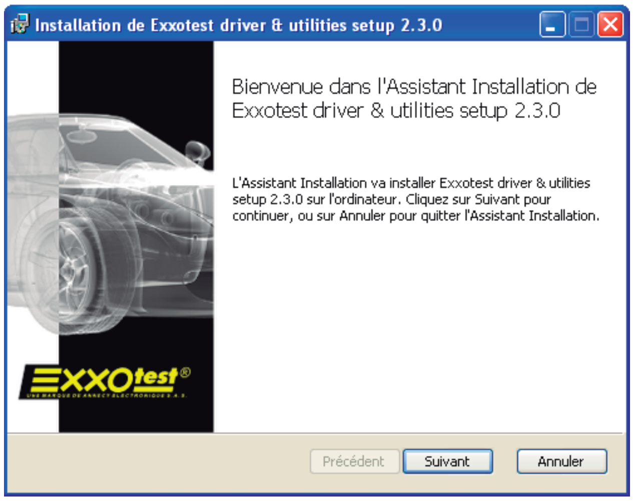
Below is an excerpt from the **official Exxotest USB-MUXDIAG-II user guide** [20] where the steps to install the necessary drivers for the correct operation of the hardware are explained in detail step by step. If we do not install these drivers, the PC will not be able to recognize the device.

5.3.4. Execution of the installation file

Step 0 : Place the installation CD that came with your hardware in the CD drive of your computer, select the « Drivers » page and launch the installation of the « **Exxotest_MUX_driver_kit_2.x.x** » file or visit the download area of the www.exxotest.com website to download and execute this file's latest version.

Step 1 : Starting the drivers installation

We recommend you at this step to check that no USB EXXOTEST® is connected to your PC
.

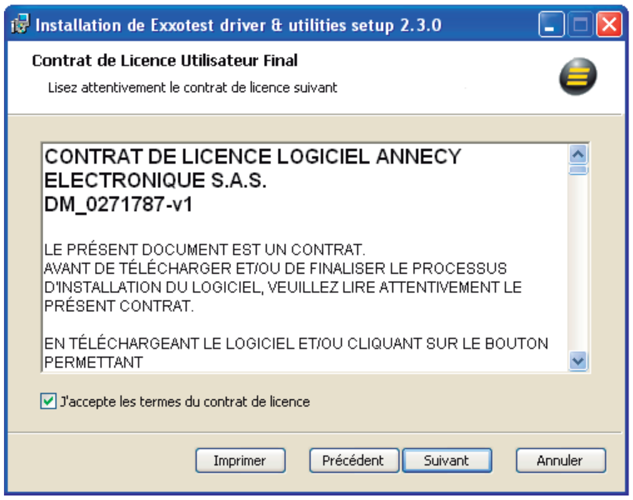

Installation de Exxotest driver & utilities setup 2.3.0

Bienvenue dans l'Assistant Installation de Exxotest driver & utilities setup 2.3.0

L'Assistant Installation va installer Exxotest driver & utilities setup 2.3.0 sur l'ordinateur. Cliquez sur Suivant pour continuer, ou sur Annuler pour quitter l'Assistant Installation.

Click on « Next ».

Step 2 : Final user license contract agreement


Installation de Exxotest driver & utilities setup 2.3.0

Contrat de Licence Utilisateur Final

Lisez attentivement le contrat de licence suivant

CONTRAT DE LICENCE LOGICIEL ANNECY ELECTRONIQUE S.A.S.
DM_0271787-v1

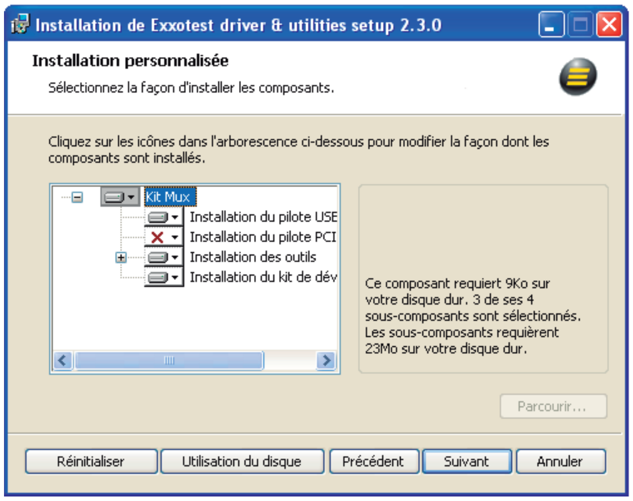
LE PRÉSENT DOCUMENT EST UN CONTRAT.
AVANT DE TÉLÉCHARGER ET/OU DE FINALISER LE PROCESSUS D'INSTALLATION DU LOGICIEL, VEUILLEZ LIRE ATTENTIVEMENT LE PRÉSENT CONTRAT.

EN TÉLÉCHARGEANT LE LOGICIEL ET/OU CLIQUANT SUR LE BOUTON PERMETTANT

☑ J'accepte les termes du contrat de licence

After reading of the license contract, tick « I accept » and click on « next » to continue the procedure.

Step 3 : Installation options selection


Installation de Exxotest driver & utilities setup 2.3.0

Installation personnalisée

Sélectionnez la façon d'installer les composants.

Cliquez sur les icônes dans l'arborescence ci-dessous pour modifier la façon dont les composants sont installés.

Kit Mux
Installation du pilote USB
Installation du pilote PCI
Installation des outils
Installation du kit de dév

Ce composant requiert 9Ko sur votre disque dur. 3 de ses 4 sous-composants sont sélectionnés. Les sous-composants requièrent 23Mo sur votre disque dur.

Parcourir...

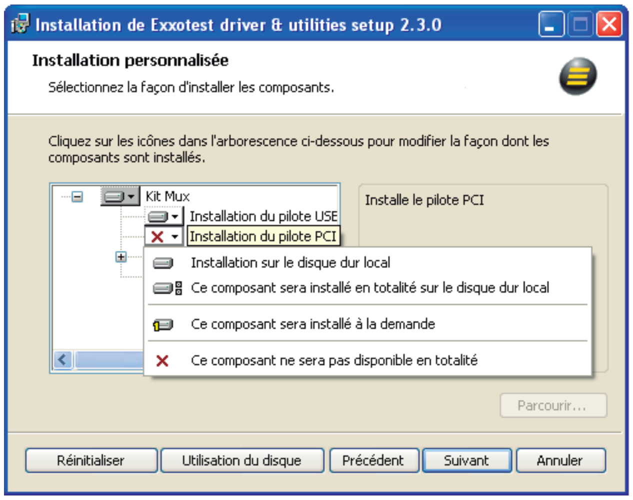
Réinitialiser | Utilisation du disque | Précédent | Suivant | Annuler

Select or unselect the options to be installed accordingly to your needs.

We therefore recommend you to keep the default configuration.
Click on « Next » to continue.

WARNING: PCI cards users, the PCI driver installation is not activated in the default configuration. WE then recommend you to proceed as described here below.

Etape 3 bis : PCI card users only


Installation de Exxotest driver & utilities setup 2.3.0

Installation personnalisée

Sélectionnez la façon d'installer les composants.

Cliquez sur les icônes dans l'arborescence ci-dessous pour modifier la façon dont les composants sont installés.

Kit Mux
Installation du pilote USB
Installation du pilote PCI

Installe le pilote PCI

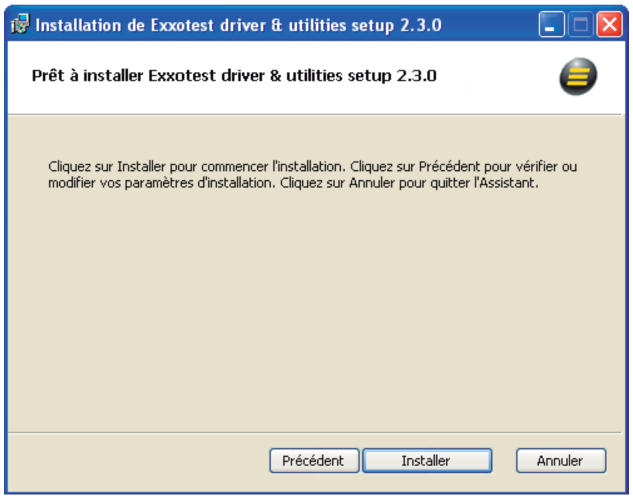
Installation sur le disque dur local
Ce composant sera installé en totalité sur le disque dur local
Ce composant sera installé à la demande
Ce composant ne sera pas disponible en totalité

Parcourir...

Réinitialiser | Utilisation du disque | Précédent | Suivant | Annuler

Click on the button in front of "PCI driver installation", select the option and click on "Next" to continue.

<u>Step 4 :</u> Starting the installation

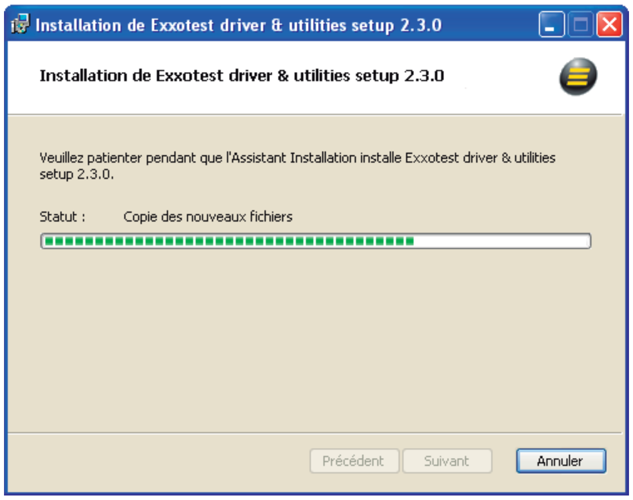


Prêt à installer Exxotest driver & utilities setup 2.3.0

Cliquez sur Installer pour commencer l'installation. Cliquez sur Précédent pour vérifier ou modifier vos paramètres d'installation. Cliquez sur Annuler pour quitter l'Assistant.

Click on "next » to start the installation as configured previously.

<u>Step 5 :</u> Installation



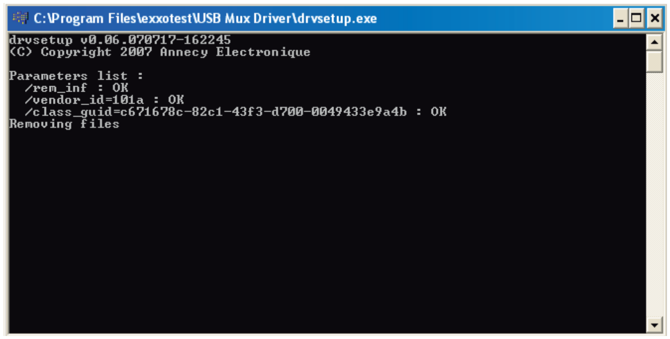
Installation de Exxotest driver & utilities setup 2.3.0

Veuillez patienter pendant que l'Assistant Installation installe Exxotest driver & utilities setup 2.3.0.

Statut : Copie des nouveaux fichiers

Installation in progress, no action from your part is required.

<u>Note</u>: the status indicator may, in certain operations stand still for several minutes.

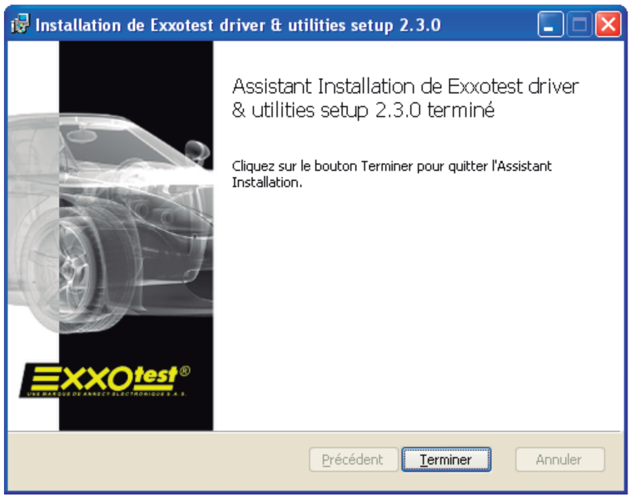


```
drvsetup v0.06.070717-162245
(C) Copyright 2007 Annecy Electronique

Parameters list :
  /rem_inf : OK
  /vendor_id=101a : OK
  /class_guid=c671678c-82c1-43f3-d700-0049433e9a4b : OK
Removing files
```

Deleting of oldest drivers found on your PC

No action from your part is required.

<u>Etape 6 :</u> End of installation



Assistant Installation de Exxotest driver & utilities setup 2.3.0 terminé

Cliquez sur le bouton Terminer pour quitter l'Assistant Installation.

Click on « Finish » to end the installation.

You can now connect your interface(s) to the PC, they will be automatically detected and installed.

C.2 CH340C

When we connect our PCB to the PC through the USB type B port implemented for it, we find that the PC does not recognize the device to which it is connected as we can see in Figure C.1.

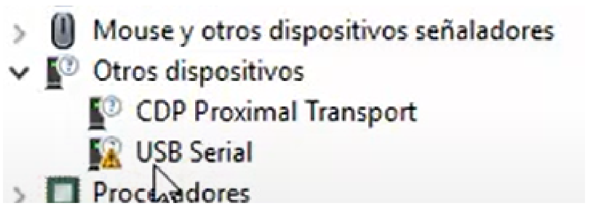

Mouse y otros dispositivos señaladores
Otros dispositivos
CDP Proximal Transport
USB Serial
Procesadores

Figure C.1 – *CH340C not recognized by the PC*

First of all, what we need to figure out is which device we are actually connecting the PC to when we connect the cable to our board. If we look at the schematic, we see that the USB type B port is directly connected to the CH340C integrated circuit, whose function is to transform the USB signal into serial communication. In order for our PC to correctly recognize our product, we must install the IC CH340C driver software. For this we follow the following steps:

1. **Downloading the correct drivers:** On the internet there are many websites where we can download drivers for a wide variety of devices, but we must make sure we download the correct drivers for our specific case. In this case we have resorted to the following website: [43].

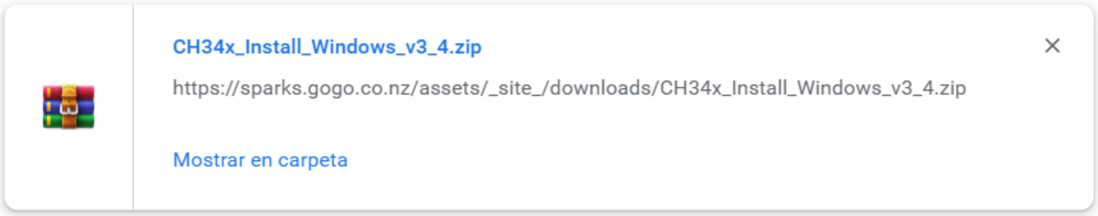
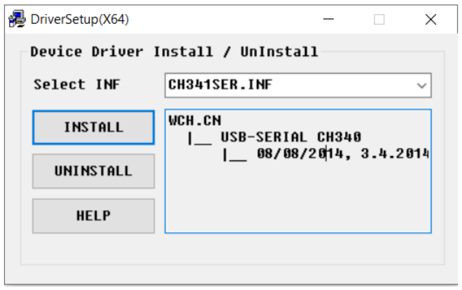

CH34x_Install_Windows_v3_4.zip
https://sparks.gogo.co.nz/assets/_site_/downloads/CH34x_Install_Windows_v3_4.zip
Mostrar en carpeta

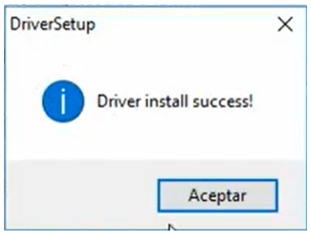
Figure C.2 – *CH340C drivers downloaded*

2. **Installing drivers:** Once we have downloaded the drivers on the PC, we must unzip the .zip file with any uncompressing software and run the installer.


DriverSetup(X64)
Device Driver Install / UnInstall
Select INF CH341SER.INF
INSTALL
UNINSTALL
HELP

```
WCH.CN
  |__ USB-SERIAL CH340
      |__ 08/08/2014, 3.4.2014
```

(a) *Mini-USB type B pinout*


DriverSetup
Driver install success!
Aceptar

(b) *Mini-USB type B schematic symbol*

Figure C.3 – *Mini-USB type B pinout and schematic symbol*

Appendix D

GranaSAT Part Manager

In the GranaSAT laboratory facilities, there is a large number of electronic components of different types classified in many compartments according to different parameters. Sometimes, it can be really difficult to know if the specific component we need is available in the laboratory, which would save us having to buy it again.

That is why GranaSAT has a web domain in which a repository has been created where we can easily search for any component we want, immediately knowing if it is physically in the laboratory. If so, we can access very useful information about the component.

In addition, each member of the GranaSAT team has credentials through which they can access the part manager with certain permissions that allow them, among other things, to update the repository by adding new components and modifying existing ones.

In Figure D.3 we can see an overview of the part manager interface.

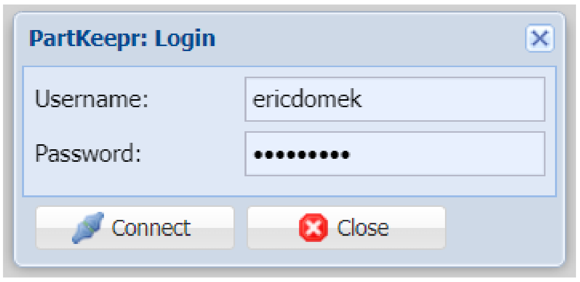
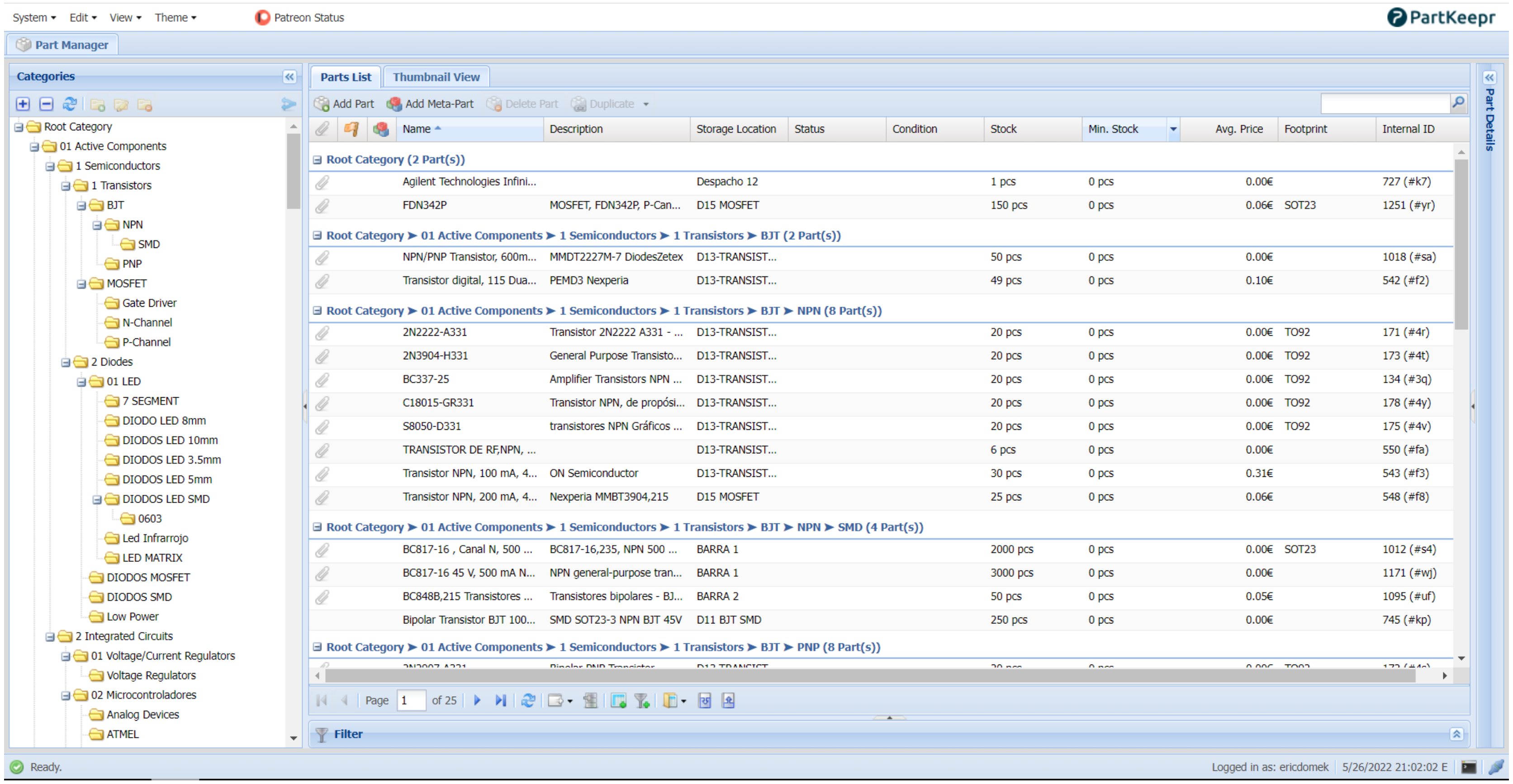


Figure D.1 – *GranaSAT part manager login window*

The Figure D.2 shows an example of the information that we can access if the component we are looking for is available in the laboratory. The specific example shows the information corresponding to IC 74HC595, a component used in the shift register module explained in subsubsection 4.1.1.6, purchased for this project and personally added to the repository.

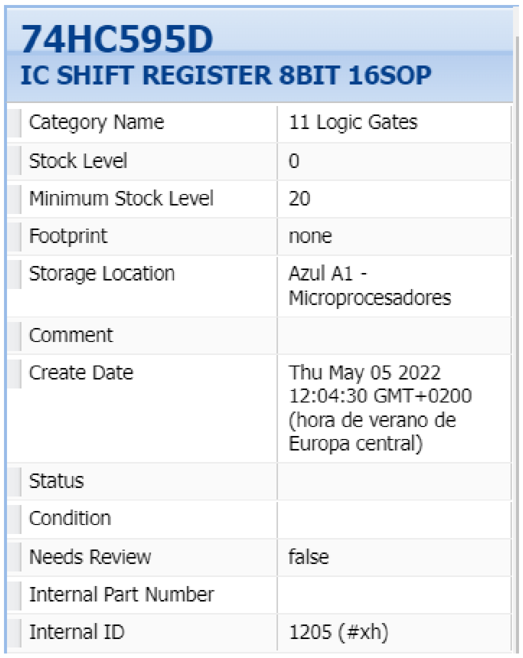
As we can see, the most relevant data about the component are the model, the quantity available, the place in the laboratory where it is stored and the package.



- Root Category
  - 01 Active Components
    - 1 Semiconductors
      - 1 Transistors
        - BJT
          - NPN
            - SMD
          - PNP
        - MOSFET
          - Gate Driver
          - N-Channel
          - P-Channel
      - 2 Diodes
        - 01 LED
          - 7 SEGMENT
          - DIODO LED 8mm
          - DIODOS LED 10mm
          - DIODOS LED 3.5mm
          - DIODOS LED 5mm
          - DIODOS LED SMD
            - 0603
          - Led Infrarrojo
          - LED MATRIX
        - DIODOS MOSFET
        - DIODOS SMD
        - Low Power
    - 2 Integrated Circuits
      - 01 Voltage/Current Regulators
        - Voltage Regulators
      - 02 Microcontroladores
        - Analog Devices
        - ATMEL

Name	Description	Storage Location	Status	Condition	Stock	Min. Stock	Avg. Price	Footprint	Internal ID
Root Category (2 Part(s))									
Agilent Technologies Infini...		Despacho 12			1 pcs	0 pcs	0.00€		727 (#k7)
FDN342P	MOSFET, FDN342P, P-Can...	D15 MOSFET			150 pcs	0 pcs	0.06€	SOT23	1251 (#yr)
Root Category ➤ 01 Active Components ➤ 1 Semiconductors ➤ 1 Transistors ➤ BJT (2 Part(s))									
NPN/PNP Transistor, 600m...	MMDT2227M-7 DiodesZetex	D13-TRANSIST...			50 pcs	0 pcs	0.00€		1018 (#sa)
Transistor digital, 115 Dua...	PEMD3 Nexperia	D13-TRANSIST...			49 pcs	0 pcs	0.10€		542 (#f2)
Root Category ➤ 01 Active Components ➤ 1 Semiconductors ➤ 1 Transistors ➤ BJT ➤ NPN (8 Part(s))									
2N2222-A331	Transistor 2N2222 A331 - ...	D13-TRANSIST...			20 pcs	0 pcs	0.00€	TO92	171 (#4r)
2N3904-H331	General Purpose Transisto...	D13-TRANSIST...			20 pcs	0 pcs	0.00€	TO92	173 (#4t)
BC337-25	Amplifier Transistors NPN ...	D13-TRANSIST...			20 pcs	0 pcs	0.00€	TO92	134 (#3q)
C18015-GR331	Transistor NPN, de propósi...	D13-TRANSIST...			20 pcs	0 pcs	0.00€	TO92	178 (#4y)
S8050-D331	transistores NPN Gráficos ...	D13-TRANSIST...			20 pcs	0 pcs	0.00€	TO92	175 (#4v)
TRANSISTOR DE RF,NPN, ...		D13-TRANSIST...			6 pcs	0 pcs	0.00€		550 (#fa)
Transistor NPN, 100 mA, 4...	ON Semiconductor	D13-TRANSIST...			30 pcs	0 pcs	0.31€		543 (#f3)
Transistor NPN, 200 mA, 4...	Nexperia MMBT3904,215	D15 MOSFET			25 pcs	0 pcs	0.06€		548 (#f8)
Root Category ➤ 01 Active Components ➤ 1 Semiconductors ➤ 1 Transistors ➤ BJT ➤ NPN ➤ SMD (4 Part(s))									
BC817-16 , Canal N, 500 ...	BC817-16,235, NPN 500 ...	BARRA 1			2000 pcs	0 pcs	0.00€	SOT23	1012 (#s4)
BC817-16 45 V, 500 mA N...	NPN general-purpose tran...	BARRA 1			3000 pcs	0 pcs	0.00€		1171 (#wj)
BC848B,215 Transistores ...	Transistores bipolares - BJ...	BARRA 2			50 pcs	0 pcs	0.05€		1095 (#uf)
Bipolar Transistor BJT 100...	SMD SOT23-3 NPN BJT 45V	D11 BJT SMD			250 pcs	0 pcs	0.00€		745 (#kp)
Root Category ➤ 01 Active Components ➤ 1 Semiconductors ➤ 1 Transistors ➤ BJT ➤ PNP (8 Part(s))									

Figure D.3 – *GranaSAT part manager general interface view*

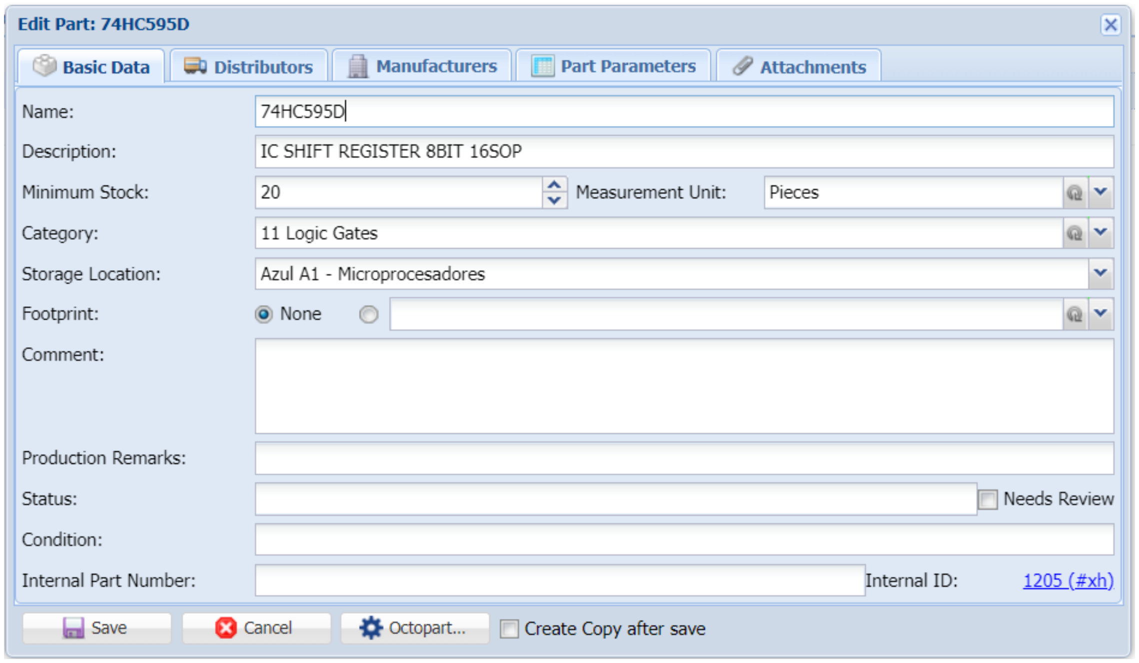


74HC595D
IC SHIFT REGISTER 8BIT 16SOP

Category Name	11 Logic Gates
Stock Level	0
Minimum Stock Level	20
Footprint	none
Storage Location	Azul A1 - Microprocesadores
Comment	
Create Date	Thu May 05 2022 12:04:30 GMT+0200 (hora de verano de Europa central)
Status	
Condition	
Needs Review	false
Internal Part Number	
Internal ID	1205 (#xh)

Figure D.2 – *GranaSAT part manager part information section*

Finally, Figure D.4 shows the dialog window that appears when we are going to edit/add a new component is displayed. We just have to specify the name, a brief description, the amount that we are going to add and the type of encapsulation of the component. It is highly recommended to add, whenever possible, an image of the component to avoid mistakes when searching for it.



Name:	74HC595D
Description:	IC SHIFT REGISTER 8BIT 16SOP
Minimum Stock:	20
Measurement Unit:	Pieces
Category:	11 Logic Gates
Storage Location:	Azul A1 - Microprocesadores
Footprint:	[✓ None]
Comment:	
Production Remarks:	
Status:	
Needs Review	☐
Condition:	
Internal Part Number:	
Internal ID:	1205 (#xh)
Create Copy after save	☐

Figure D.4 – *GranaSAT part manager edit window*

Appendix E

Project budget

E.1 Power budget

In this section we will carry out the calculation of the total power consumption of the product. On many occasions, the component Datasheet indicates three values to mark their power consumption: minimum, typical and maximum. That is why we will carry out the power calculations taking into account the three possible situations. In this way we will obtain a better case, when the product will consume less power; a worst case, when the product will consume more power; and a typical case, which will indicate the most probable consumption.

We cannot forget that in the consumption calculations we must also take into account the consumption of the headlights, not only that of the components of the plate. In fact, this consumption will be the dominant one.

E.1.1 Headlights power consumption

In this section, a study of the power consumption of each function has been carried out separately for each headlight.

Rear pilots power budget

Function	**Pilots**			
	Trunk Q8	**Fender Q8**	**Trunk Lamp Q5**	**Trunk LHD Q30**
Brake (A)	0,34	0,4	0,28	0,22
TI (A)	0,3	0,25	0,51	0,29
Position (A)	0,54	0,26	0,37	0,07
Back (A)	0,79	-	-	0,2
Quantity	1	2	6	1
Max. consumption (A)	1,97	1,82	6,96	0,78
BUDGET (A)	**11,53**			

Table E.1 – *Rear lights power budget*

The power calculations will be made taking into account the consumption when all the functions are on for the same time. We must do it this way, since, if our product offers the possibility of turning on all the headlights at the same time, we must make it support, at a minimum, the required power consumption for that to happen.

Table E.1 displays the power budget for all taillights as a whole. We have decided to separate this quote from that of the headlight because both have very different functions to take into account.

Headlight power budget	
Function	**Pilot**
	Headlight Renault Megane 2016
DRL (A)	1,36
LB (A)	3,9
HB (A)	3,9
TI (A)	-
BUDGET (A)	**9,16**

Table E.2 – *Headlight power budget*

Now we will move on to calculating the power budget of the rest of the PCB components: microcontrollers, ICs, LEDs, etc.

Components power budget						
Type of consumption	**Components**					
	LEDs	**ATA6662C**	**CH340C**	**DF-Player**	**74HC595**	**Arduino UNO**
Typical (A)	0,03	0,0012	0,012	0,02	0,000001	0,041
Minimum (A)	0,015	0,00002	0,012	0,02	0,0000001	0,041
Maximum (A)	0,03	0,002	0,03	0,02	0,000001	0,041
Quantity	45	1	1	1	4	1
Typ. consumption (A)	1,35	0,0012	0,012	0,02	0,000004	0,041
Min. consumption (A)	0,45	0,0002	0,012	0,02	0,0000004	0,041
Max. consumption (A)	1,35	0,002	0,03	0,02	0,000004	0,041
TYP. BUDGET (A)	**1,4242**					
MAX. BUDGET (A)	**1,443**					
MIN. BUDGET (A)	**0,5232**					

Table E.3 – *PCB components power budget*

We can notice that practically all the consumption due to the components of the board is due to the LEDs, due to their large number and the low consumption of the other ICs

E.2 Economic budget

This section of the appendix will analyse the investment made in terms of costs of material and manpower that has been required in order to perform this bachelor's thesis. This estimation should give an idea of how much it could cost to carry out the same project in an actual engineering company. To be as realistic as possible, we will take into account all the physical materials used for each part, plus the cost of software and engineer labor. In this way the estimate will be more precise.

Given the nature of the work, we will divide the budget into 4 parts:

- Cost of the complete set of pilots corresponding to different vehicles.
- Reverse Engineering by LIN bus sniffing
- Design and implementation of the PCB necessary for the control of the pilots.
- Design and implementation of the demonstrator support (wooden plank).

In the first place in Table E.4, we will make an estimate of the cost of the complete set of different pilots that have been necessary and have made it possible to carry out all the phases of the project, especially since their control is part of its main objective. The prices of the pilots have been obtained from different authorized dealers of spare parts for vehicles, as well as from second-hand sales websites.

Pilot set			
Item	**Cost [€]**	**Quantity**	**Total cost [€]**
Trunk Light RH Audi Q5 2020	239, 58 [33]	3	718, 74
Trunk Light LH Audi Q5 2020	239, 58 [33]	3	718, 74
Trunk LHD Infiniti Q30 2015	117, 67 [6]	1	117, 67
Headlight Renault Megane 2016	276 [53]	1	276
Fender Left Audi Q8 2018	296, 59 [54]	1	296, 59
Fender Right Audi Q8 2018	296, 59 [54]	1	296, 59
Trunk Light Audi Q8 2018	785, 06 [55]	1	785, 06
BUDGET			3.209, 39

Table E.4 – *Pilot set expenses*

Now we continue in Table E.5 with the estimate of the budget to carry out the reverse engineering task to know the details of the LIN communication that the vehicle maintains with the headlights in a normal way. Notice that the license to use the "*AUDI Rearlamps management v1.1.13*" software is free of charge as it has been provided by the company itself for educational purposes.

Reverse engineering - materials	
Item	**Total cost [€]**
EXXOTEST USB-MUXDIAG-II	515, 82 [56]
Logic analyzer Saleae, 24 MHz, 8 channels	15, 49 [49]
AMUX-2C2L adapter	4, 95 [8]
DB9 to 3 pin adapter	3, 95 [9]
Logic 2.3.39 license	0
AUDI Rearlamps Management v1.1.13 software license	0
BUDGET	540, 21

Table E.5 – *Reverse engineering by LIN bus sniffing necessary materials expenses*

It should be noted that some components are sold wholesale and, although for our application we would have needed fewer units, we have been forced to purchase the packs available at the distributor. Nevertheless, sometimes, buying components in bulk is much more profitable than buying them separately, especially since the surplus units are stored in the GranaSAT laboratory and registered in its **part manager**, whose operation is explained in Appendix D. In any case, the quantity shown in the table corresponds to the number of components used in the board in order to better illustrate the real cost of the board.

PCB components			
Item	**Cost [€]**	**Quantity**	**Total cost [€]**
Arduino UNO	26, 99 [50]	1	26, 99
ATX switching power supply	54, 99 [52]	1	54, 99
ATA6662C LIN transceiver	0, 65 [10]	1	0, 65
USB type B	2, 20 [11]	1	2, 20
CH340C USB to serial	0, 50 [12]	1	0, 50
74HC595 Shift Register	0, 09 [13]	4	0, 36
DF-Player mini	0, 97 [14]	1	0, 97
FDN342P P-Channel MOSFET	0, 02 [15]	33	0, 82
2N7002 N-Channel MOSFET	0, 01 [16]	33	0, 41
VS1838 IR Receiver	0, 06 [17]	1	0, 06
SSD1306 OLED display	7, 58 [51]	1	7, 58
Phoenix Contact Terminal Blocks	0, 13 [18]	19	2, 47
SMD 0805 resistors	0, 08	106	8, 48
SMD 0805 capacitors	0, 08	7	0, 56
SMD 0603 LEDs	0, 15	37	5, 55
1N4148 diodes	0, 01	2	0, 02
BUDGET			112, 61

Table E.6 – *Components present in the PCB expenses*

Once we know the budget for the components that will be soldered to our PCB, we will show the price that its manufacture has cost. Let us remember that, due to lack of time and technology, we decided to order the manufacture of the PCB in China through the website `www.jlcpcb.com`. This manufacturer calculates the price of their PCBs based on their dimensions, materials, finishes, etc. The characteristics of our PCB can be consulted in Table 4.14. It should be noted that, as was the case with the components, this manufacturer sells its products wholesale so, for our application, the purchase may not be very profitable, since we only need one PCB, and the minimum order allowed is of 5. Also, coming from China, shipping costs will be high. Table E.7 shows a breakdown of plate manufacturing costs.

PCB manufacturing	
Product	**Cost [€]**
Engineering fee	3,80
Board (5 uds.)	4,56
Shipping	25,41
Stencil	6,65
BUDGET	40,42

Table E.7 – *PCB manufacturing expenses*

Table E.8 shows a balance on the costs of the material involved in the design and implementation of the wooden plank that will serve as a support to carry out the demonstration of the operation of the headlights.

Wooden plank			
Object	**Cost [€]**	**Quantity**	**Total cost [€]**
M5 nuts	0,07	36	2,52
M5 washers	0,06	36	2,16
M5 threaded unions	0,24	18	4,32
M5 threaded rods	0,44	18	7,92
M6 threaded unions	0,44	18	7,92
M6 threaded rods	0,44	11	7,92
M6 washers	0,06	36	2,16
M6 nuts	0,07	36	2,52
244 x 122 cm wooden plank	20,99	1	20,99
BUDGET			58,43

Table E.8 – *Wooden plank fabrication expenses*

Table E.9 shows a count of budget dedicated to software that has been required during the project. It can be seen that none of the programs used has involved any cost, since they are either open source or the license has been obtained through sponsorship by the GranaSAT laboratory.

Software licenses		
Software	**License owner**	**Cost [€]**
AUDI Rearlamps Management v1.1.13	GranaSAT	Free (Sponsorship)
Logic 2.3.39	Eric Domek	Free license
Altium Designer® 21	GranaSAT	Free (sponsorship)
SolidWorks®	GranaSAT	Free (sponsorship)
Arduino IDE	Eric Domek	Free license
LT-Spice	Eric Domek	Free license
Overleaf	Eric Domek	Free license
BUDGET		$0,00$

Table E.9 – *Software licenses expenses*

Human resources		
Staff	**Time [h]**	**Cost [€]**
Junior engineer	960	$9.600,00$
Senior engineer	64	$3.200,00$
BUDGET		$12.800,00$

Table E.10 – *Human resources expenses*

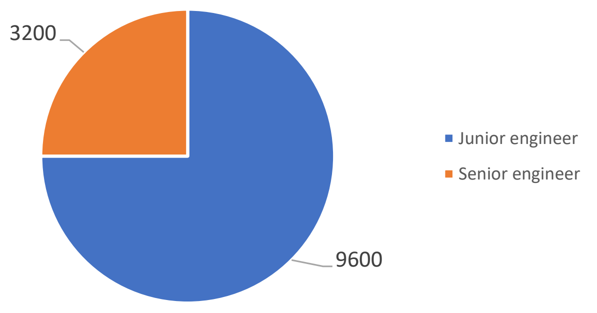
Regarding the budget corresponding to human labor, we have concluded that a real company would have needed to hire a junior engineer, who would do most of the work, and a senior engineer, who would supervise weekly the work done by the junior engineer. We have assigned the junior engineer a salary of €10/hour, while the senior engineer would receive €50/hour. Making estimates based on the Gantt chart available for consultation in section 3.5, we have obtained the budget present in Table E.10 for the human resources of the project.

Table E.11 displays the final budget by compiling the budget dedicated to each section of the project.

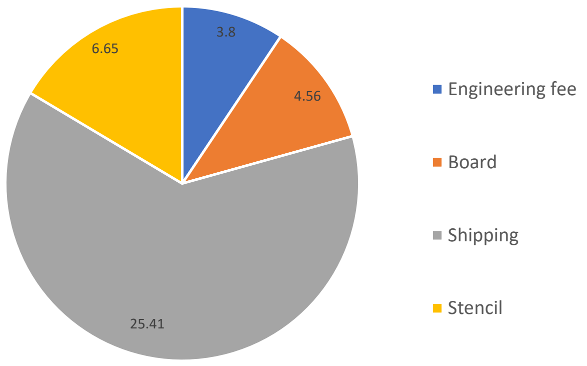
TOTAL BUDGET	
Section	**Cost [€]**
Pilot set	$3.209,00$
Reverse engineering - materials	$540,21$
PCB components	$112,61$
PCB manufacturing	$40,42$
Wooden plank	$58,43$
Software licenses	$0,00$
BUDGET	$3.960,67$

Table E.11 – *Total budget*

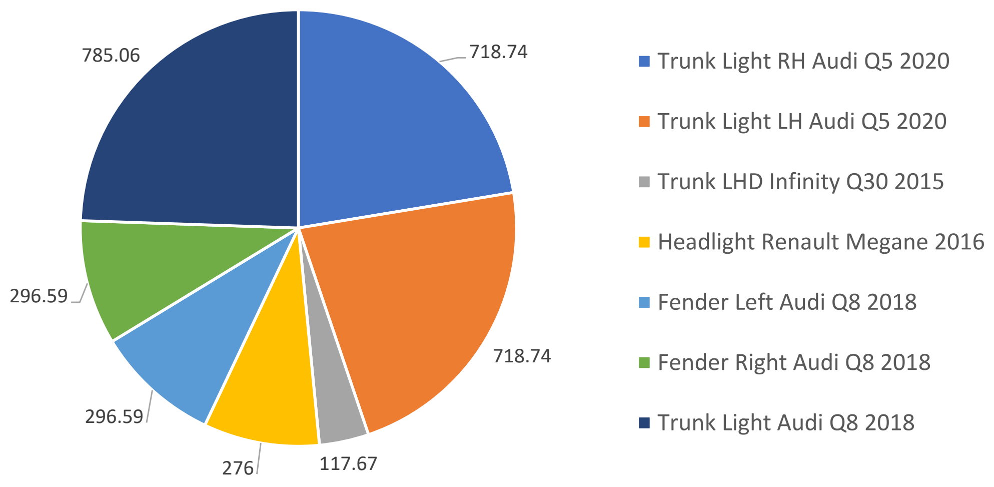
Figure E.2 shows in a very illustrator graphic the distribution of the budget corresponding to the different sections of the project. As can be seen, the vast majority of the money has been dedicated to human resources and the set of pilots corresponding to different vehicles (Audi Q8, Audi Q5, Renault Megane and Infinity Q30) whose budget breakdown can be seen at Figure E.1 (c). In Figure E.1 (a) we can see the budget distribution for human resources. As a curiosity, we can highlight that the ratio between hours worked and salary is infinitely more beneficial in the case of the senior engineer. In the case of the PCB manufacturing costs present in Figure E.1 (b), we can see that, as we are not interested in a bulk order, the profitability of the purchase is greatly reduced due to the high cost of shipping.



(a) *Human resources*



(b) *PCB manufacturing*



(c) *Pilot set*

Figure E.1 – *Budget allocations statistics [€]*

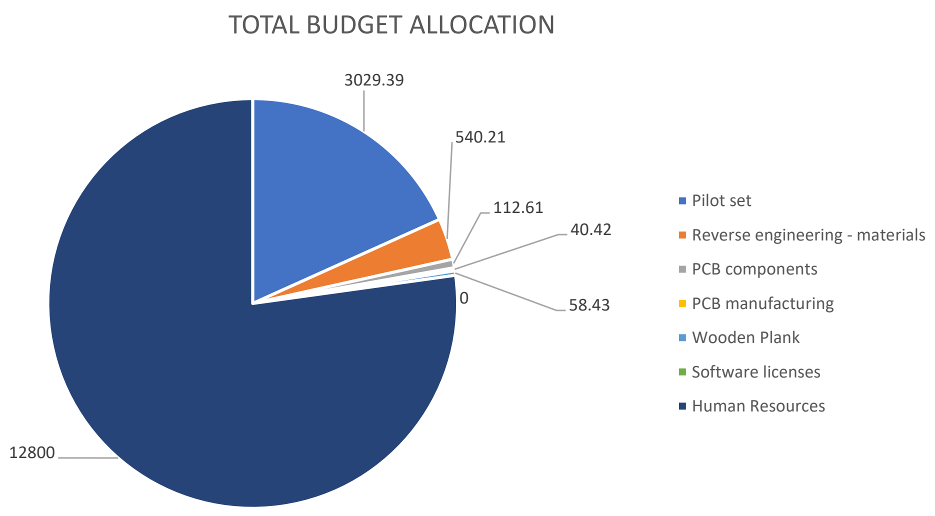


Figure E.2 – *Final budget allocation*