Summary

This project is an example of the evolution of know-how in ETSEIB Motorsport. It joins CAT07e's telemetry with CAT08e's data acquisition.

This document summarizes the whole process of design, construction and verification of a Telemetry ECU for ETSEIB Motorsport Formula Student CAT09e. Moreover, it describes the development of an Android APP to read and manage real-time data.

This memory includes both hardware and software final choices, going through each component definition and functionality.

It aims to be useful to know car reactions to different inputs and get its best performance on track.

ETSEIB

Index

ETSEIB

Table List

ETSEIB

Figure List

ETSEIB

1. Glossary

The acronyms used in this memory and their meanings are listed below:

ECU – Electronic Control Unit

PCB – Printed Circuit Board

CAN – Control Area Network

EMI – Electromagnetic Interferences

UART – Universal Asynchronous Receiver-Transmitter

TCP – Transmission Control Protocol

IP – Internet Protocol

LED – Light Emitting Diode

AP – Access Point

PTC – Positive Temperature Coefficient

SMD – Surface Mounted Device

TH – Through Hole

RAM – Random Access Memory

USB – Universal Serial Bus

PLL –Phase-Locked Loop

PDF – Portable Document Format

DHCP – Dynamic Host Configuration Protocol

SSID – Service Set Identifier

BOM – Bill Of Materials

SDK – Software Development Kit

API – Application Programming Interface

2. Preface

2.1. Project Origins

The project context is the Formula Student competition and ETSEIB Motorsport Barcelona, our school team, in which 30 students design and build a Formula Student car to compete all over the world. Each university participating in this competition creates its own racing car and travels through the world to race it during summer. The competition includes two types of tests, static and dynamic. In the first ones, students have to defense their knowledge in cost analysis, business plans and design whereas dynamic events such as accelerations or endurance are placed in the other.

ETSEIB Motorsport was born in 2007 and one car is build each year since then. This year, the CAT09e is going to race in Italy, Germany and Catalonia.

2.2. Motivation

Nowadays, data acquisition and post-processing is getting more importance than ever in automotive and motorsport projects. The future of autonomous cars mostly depends on this data.

In motorsport, data is used to setup the car and control its status in real time.

Nowadays, the current Data Acquisition of the team only allows saving data in a USB memory pen. Once the car is stopped, the pen can be connected to the computer USB and do several procedures to watch the acquired data. This procedure consumes a great quantity of time when the car is on track, causing testing to be slow.

In order to optimize test time, this projects aims to join CAT08e’s Data Acquisition with CAT07e’s Telemetry and improve both systems.

2.3. Previous Requirements

This project is based on learning by doing and the transmission of knowledge. For this reason, the only previous requirements are reading the last projects of the team and start the season learning how to use the software used by the team. Every season, the first month is spent in these two objectives.

ETSEIB

3. Introduction

3.1. Project Objectives

The main objective of this project is to design, build and test an ECU for managing data of the Formula Student car CAT09e, from Data Acquisition to Telemetry, and create an android app to read real-time data and post process it.

This objective can be summarized with the following points:

- Join the Data Acquisition with the Telemetry in a unique ECU
- Add an additional CAN bus connection to make the ECU more versatile
- Make a more intelligible software structure in which it is easier to modify data
- Speed up testing using a mobile application that allows you to view data and make graphs.

3.2. Project Scope

The project scope is based on three points:

- Arrange and improve CAT08e's hardware: It had power problems and misconnections. Also a new CAN bus connection will be available.
- Arrange and improve CAT08e's software: Telemetry was not included and some user parameters were difficult to be modified.
- Development of an Android APP to read TCP packets via Wi-Fi from the module.

4. Background and viability analysis

4.1. Background in ETSEIB Motorsport

At the beginning of the team, a commercial Data Acquisition module was used to obtain car data. Because of the high price and the power consumption of this solution and the fact that competition judges prefer self-made components, in CAT06e the team started to use semi commercial modules such as Arduino. The following year, in CAT07e season, the team designed and built its own PCBs for the first time.

This project, as mentioned before, is based on CAT07e Telemetry and CAT08e Data Acquisition.

4.1.1. CAT07e Telemetry

During the 2013-2014 season, one team member developed a PCB to send real-time data to an iOs device. It used a Wi-Fi connection and an IP/TCP protocol. From that project, even though the PCB worked, he extracted the following conclusions:

- The position of the device, behind the pilot, prevented a good Wi-Fi signal
- There was not enough sensors in the car
- With a bidirectional connections would be easy to modify car data

4.1.2. CAT08e Data Acquisition

With the next car, the CAT08e, another team member designed a Data Acquisition PCB which included a WiFly module to implement the telemetry function but, due to the short developing time, it never worked. However, it supposed a step forward in the team's knowledge by using a PIC32 from Microchip instead of a PIC18 microcontroller. It allowed him to develop a solid and powerful Data Acquisition program.

Some of his conclusions are listed below:

- Using a PIC32 greatly improved the functionality of the ECU.
- Data was saved at the desired frequencies.
- Using DC/DC converter would increment the efficiency of the ECU

Though CAT08e's Data Acquisition gave good results, some malfunctions were detected. Due to wrong voltage input connections, the module reset from time to time. Also, due to bad connections, it was impossible to send UART messages to the WI-FI module that was already built on. To modify the data, the main code had to be modified and this was really slow and a source of problems. Finally, some errors were detected in the exported files that did not allow an accurate post processing.

4.2. Viability

4.2.1. Technical Viability

Due to the previous experience of the team and the first month of learning, the hardware design of the project did not represent a great difficulty. All the chosen components had been tested before and most of the hardware solutions too.

The difficulty comes in the software, because it is the second year using a PIC32 microcontroller with the consequent lack of experience. Moreover, an android app will be used for the first time so java developing knowledge is required.

4.2.2. Economic Viability

The ECU will be used in a prototype racing car, therefore only a few number will be assembled. It must take into account that some of the components used in the ECU have been provided by sponsors of the team, which significantly reduces costs. The tablet used as interface for data analysis is a sponsored product too.

In point 10 of this project report, the cost analysis is described extensively.

4.2.3. Environmental Viability

The environmental viability is not a limiting aspect in the development of this project, because we only manufacture a prototype. There are a set of international guidelines (such as the RoHS), which restricts the use of certain hazardous materials in the manufacture of electronic components.

Point 11 in this document exposes a detailed study of the environmental impact.

5. Specifications

Inside this chapter, the requirements for each component of this project will be specified: firstly the module specifications including both hardware and software and secondly the App specifications.

5.1. Module

The objective of the module is to read CAN bus messages from the car, codify them into a predefined specified format and send all the data to an app via Wi-Fi connection.

5.1.1. Hardware specifications

- Rated input voltage of 12V
- 2x CAN bus connections
- Wi-Fi module
- USB connection (memory pen)
- Antenna in a surface position

5.1.2. Software specifications

- Read and send data through CAN bus
- Up to 100Hz data recording in csv format
- IP/TCP protocol and AP mode
- Easily modifiable data

5.1.3. Module environment

CAT09e's electronics are composed by one main control unit (MicroAutoBox) from dSPACE, one BMS, and self-made PCBs connected to all the sensors and actuators of the car. Four CAN networks connect all the subsystems to share information.

Sensors data are sent to dSPACE, which analyzes them, makes decisions and also sends all the information through a CAN bus.

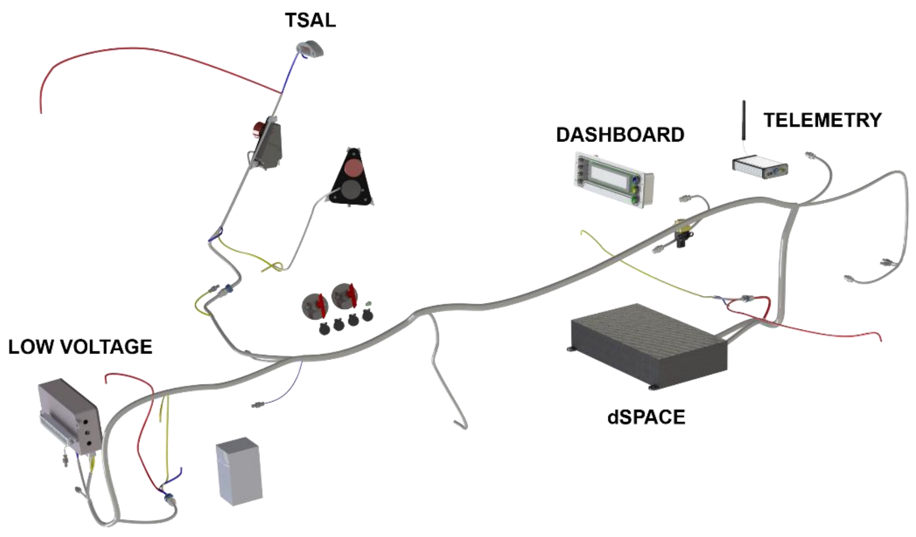


Figure 1. CAT09e electronics. Source: Own.

5.2. Android App

The main functionality of the app is to know the status of the car in real time. Moreover, it has to allow the possibility to record and display the data to provide a fast analysis in a car test session. Therefore, it must include:

- Wi-Fi connection
- File Storing
- Graphic Viewer

6. Planning

The project has a duration of 4 months. It begins on September with the start of one formula student year and ends in late December, with the aim to be ready for testing the car in February.

The project has been split into 7 phases: Study of previous projects, definition of specifications, technical viability study, design, assembly, program and test.

One of the most important values in the formula student world is the transmission of knowledge each year from the former members to the new ones. That is why every member of the team starts the season learning from the previous projects.

After that, the specifications of the product are defined and the design phase starts. This is the most complex part because all the project depends on the good design of the product more than on any of the other steps.

Once the design is done, the Assembly phase starts, in which the PCB is soldered and all the connections expected to work on the design are tested.

The last and most important part in this project is related to the software. It has to join each component of the PCB with the others. Also the android app is created in this part.

When everything is finished, the final step in the project is to check and validate all the components in a real test situation.

Fig. 2 shows a Gantt diagram with all these described phases, indicating its start and end dates, including some subtask that should be done during the project.

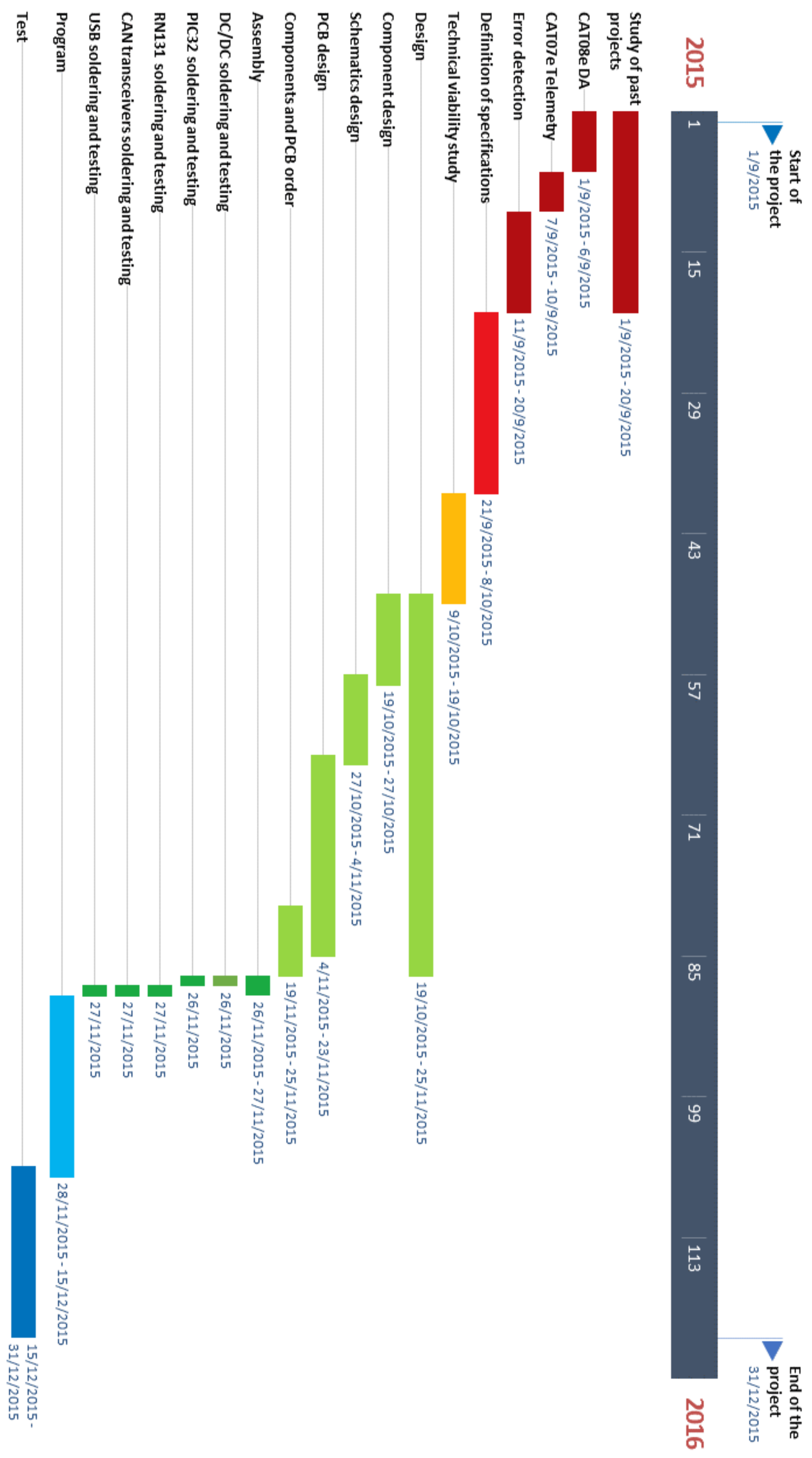


Figure 2. Gantt diagram of the project planning. Source: Own.

7. Hardware

This chapter will discuss only the hardware of the designed PCB because the analysis of the hardware of an android device is not part of this project.

7.1. General Overview

Fig. 3 shows the final PCB layout with all the components on board. Its function, as mentioned before, is to provide the team all the data from the car sensors via csv files and TCP messages.

We can divide it into 5 basic parts explained during this chapter: Connectors, Power, Control, CAN-Bus, USB and Wi-Fi.

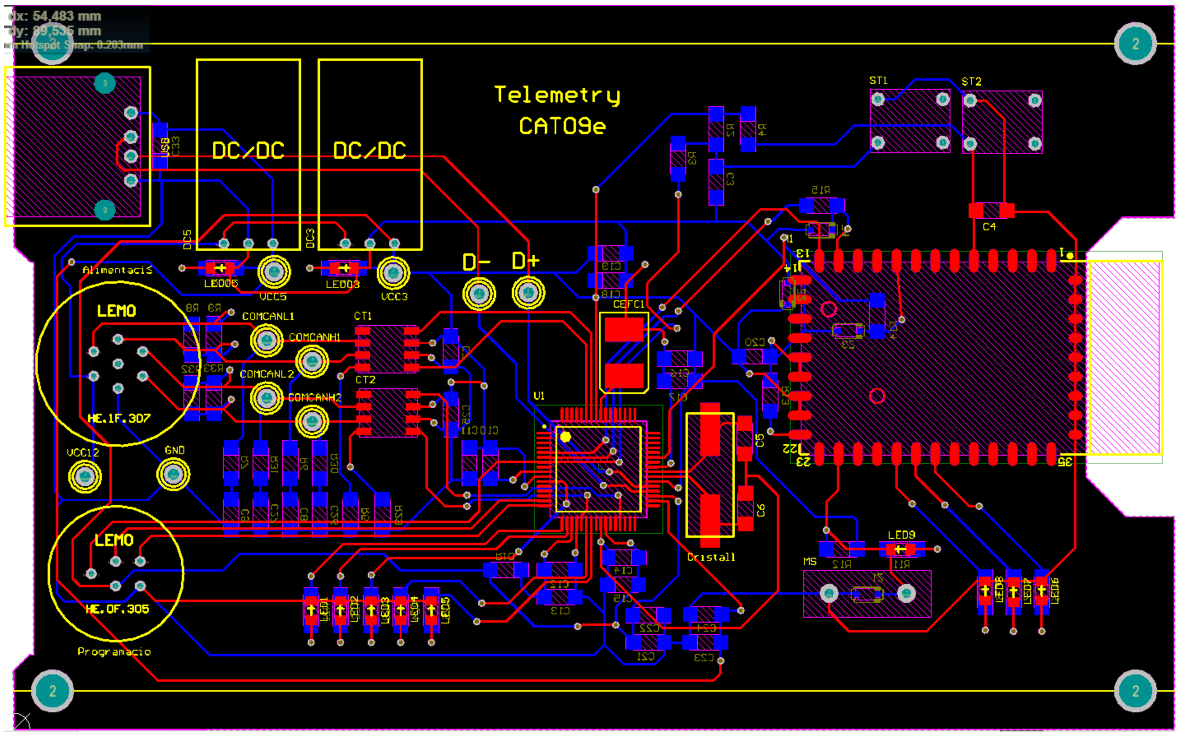


Figure 3. PCB general overview. Source: Own.

7.2. Power

To solve the power and temperature problems of the last year Data Acquisition PCB, the module will be powered by two RECOM Power DC/DC converters. One for the 3.3V components and the other for powering the USB, which needs a 5V input voltage.

These converters will reduce the consumption of the ECU at the same time that protect the circuit from overloads and short-circuits.

RECOM Power R-78B series characteristics	
Input Range	4.75 – 32 V DC
Output current	1A
Efficiency	~90%

Table 1. DC/DC characteristics. Source: Own.

Figure 4. RECOM Power R-78B DC/DC. Source: Own.

7.3. Control

To control the communications and data flow a PIC32MX795F512H is used with a maximum clock frequency of 80MHz that allows high speed data processing. In addition, it has the requirements needed for our purpose, as shown in Table 2.

Number of pins	64	Temperature range	-40 to 105 ºC
Maximum frequency	80 MHz	CAN module	2
Power voltage	2.3-3.6 V	UART module	5
Program memory (Flash)	512 KB	USB module	1
Data memory (RAM)	128KB	Timers	5

Table 2. PIC32 MX795F512H characteristics. Source: Own.

To ensure the clock precision an external 8 MHz crystal oscillator has been added to the microcontroller.

ETSEIB

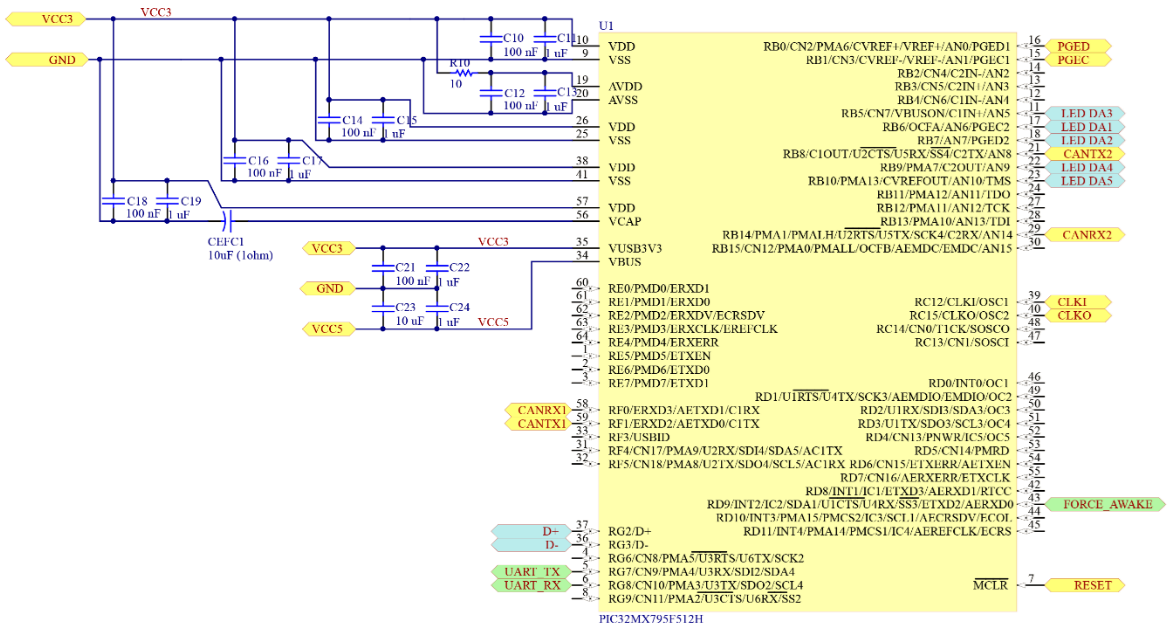


Figure 5. Control schematics. Source: Own.

7.4. CAN Bus

A controller area network (CAN bus) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer. It is a message-based protocol.

Data is transmitted without a clock signal in an asynchronous format and must contain an ID and the message data.

With the objective of making this ECU more versatile, it includes 2 CAN transceivers which allow two CAN bus connections. That can be useful to reduce the net jam and to acquire data from different networks.

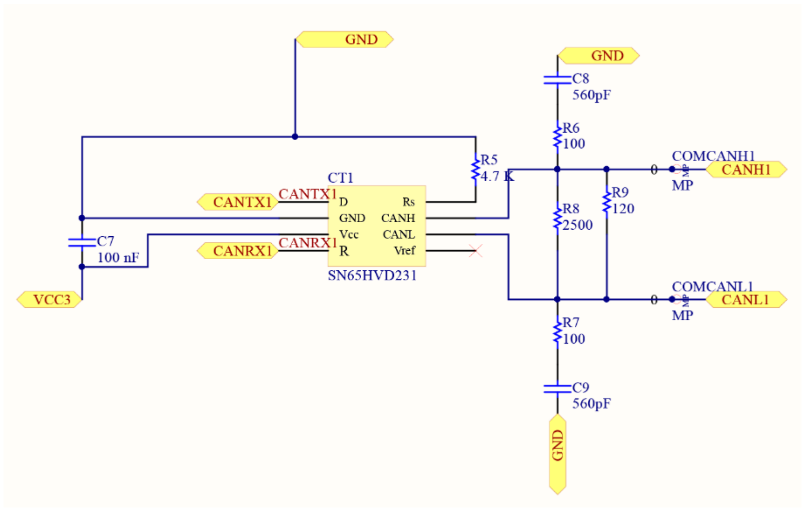


Figure 6. Schematic of one of the CAN bus connections. Source: Own.

7.5. USB

The USB (Universal Serial Bus) is a standard asynchronous protocol very extended in personal computers. It is the simplest way to connect the PIC to a mass storage because the C compiler has predefined USB 2.0 libraries.

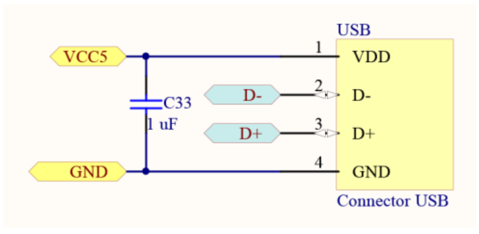


Figure 7. USB schematic to connect a memory pen. Source: Own.

7.6. Wi-Fi

The Wi-Fi connection of the module is based on the WiFly RN131C module from Microchip. It receives UART messages from the PIC and sends it via TCP.

UART (Universal Asynchronous Receiver-Transmitter) is a standard protocol to send a byte flux from a buffer via digital pulses that represent the bits.

Figure 8. WiFly module. Source: Microchip.com.

TCP (Transmission Control Protocol) is a protocol to communicate two devices and provides reliable, ordered and error-checked delivery of a stream of bytes.

To improve the range of the Wi-Fi signal an external antenna will be placed in front of the pilot.

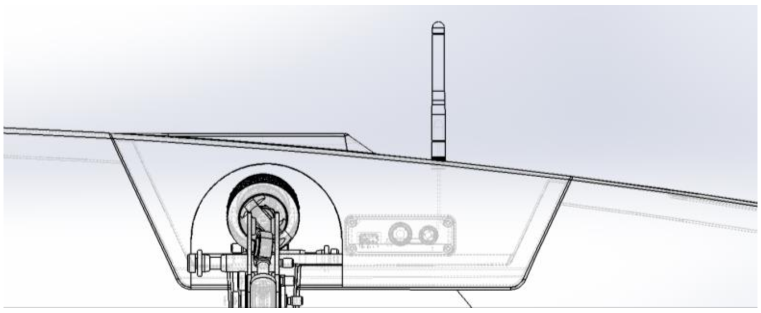


Figure 9. Antenna location in the monocoque. Source: Own.

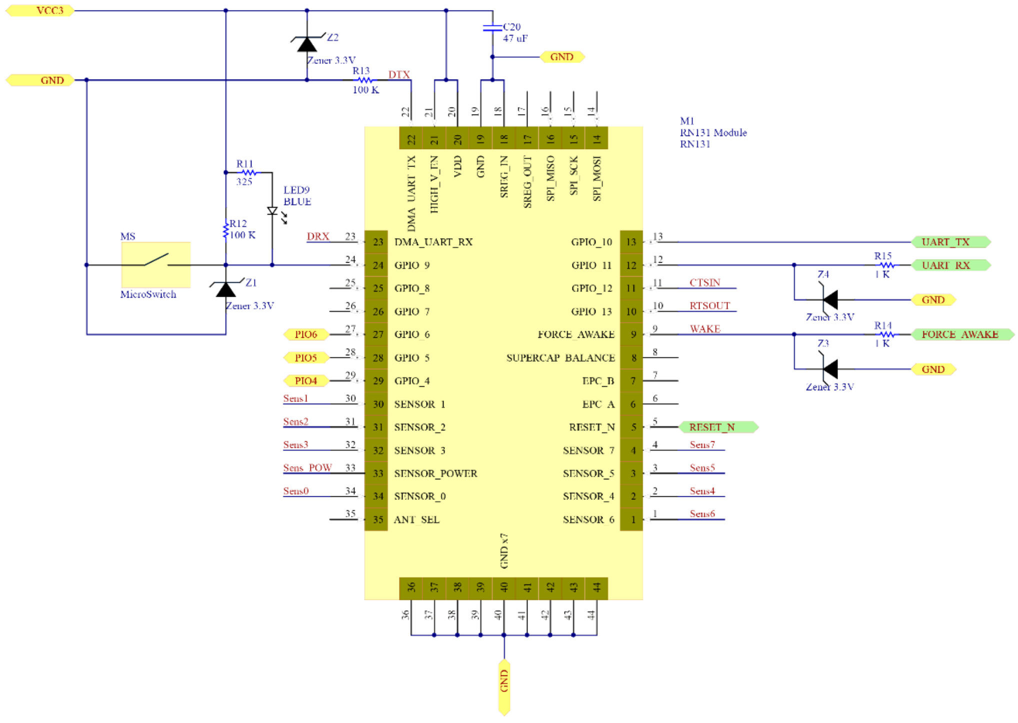


Figure 10. WiFly module schematic. Source: Own.

7.7. Complete ECU schematic

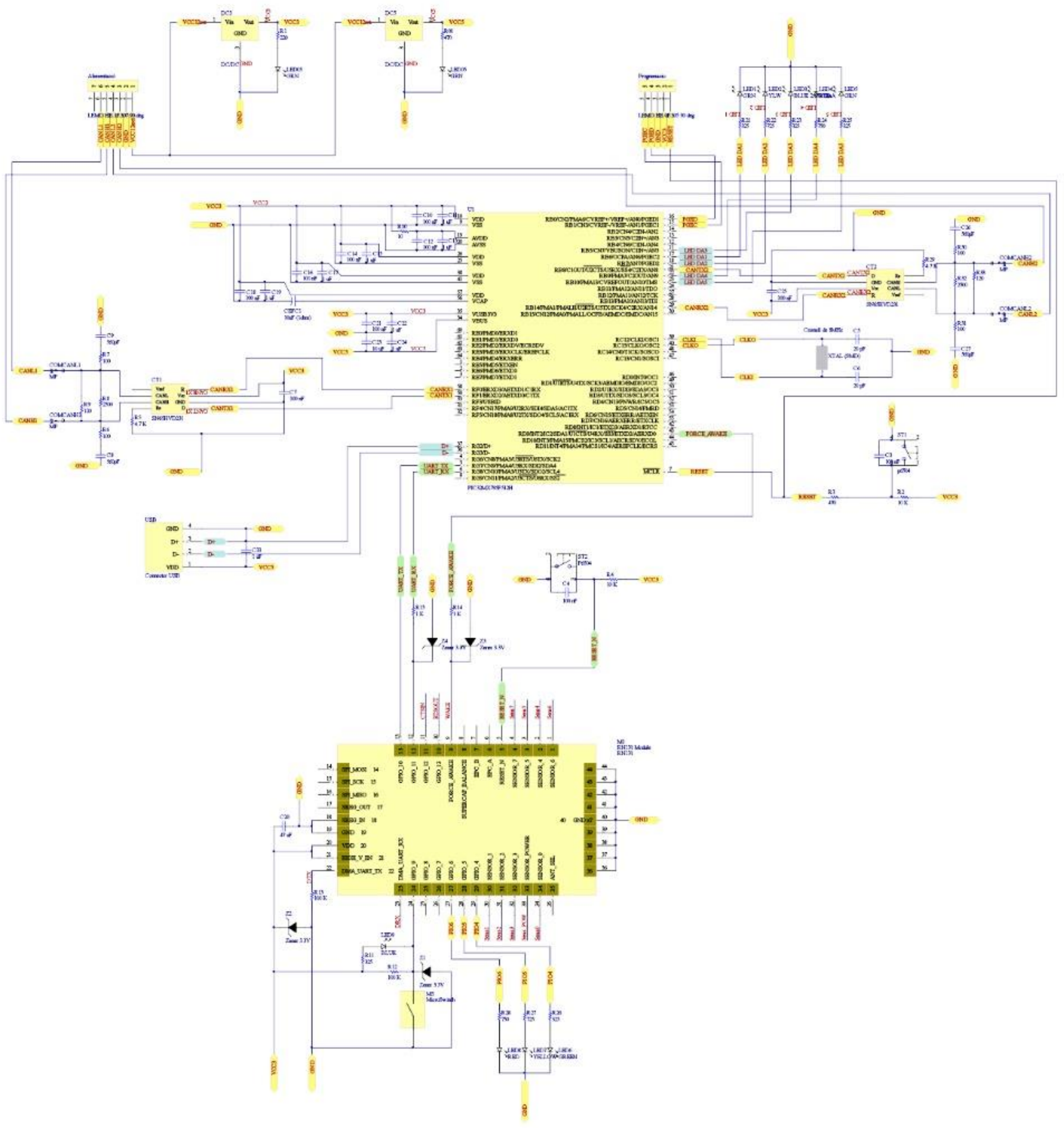


Figure 11. Complete ECU schematic

8. Software

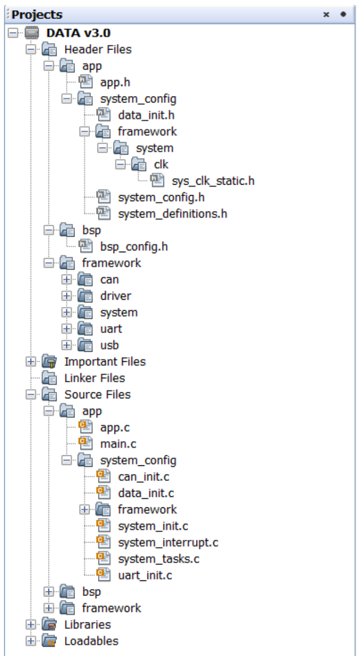
In this chapter the software, including both microcontroller and WiFly module and the Android App, is explained. As the project is an evolution of previous projects, there will be a general summary with special emphasis on their improvements.

8.1. Microcontroller software

8.1.1. C++ Language and MPLABX

C++ is a general purpose programing language with imperative, object-oriented and generic programing features. It is one of the languages used to program PIC microcontrollers using the Microchip platform MPLAB IDE. To make things easier for the programmer, Microchip developed a series of libraries and a configurator plugin called Harmony [8], which is useful to start building any PIC32 application.

8.1.2. Files structure



```
Projects
DATA v3.0
  Header Files
    app
      app.h
      system_config
        data_init.h
        framework
          system
            clk
              sys_clk_static.h
        system_config.h
        system_definitions.h
    bsp
      bsp_config.h
    framework
      can
      driver
      system
      uart
      usb
  Important Files
  Linker Files
  Source Files
    app
      app.c
      main.c
      system_config
        can_init.c
        data_init.c
        framework
        system_init.c
        system_interrupt.c
        system_tasks.c
        uart_init.c
    bsp
    framework
  Libraries
  Loadables
```

Figure 12. Files structure. Source: Own.

MPLAB Harmony projects are structured in a way that isolates the code necessary to configure the PIC32 from the library modules themselves and from your application code.

The `main.c` file contains primarily just the program loop. The application files (`app.c` and `app.h`), where the user introduces the app functions, are separated from the configuration files in the `system_config` folder. The library modules that make up the MPLAB Harmony framework, saved in the framework folder, use the definitions stored in the header `system_config.h`. It starts every peripheral, timer and interrupt that will use when called from the `Sys_initialize` function.

8.1.3. Program structure

The program starts in the main file, composed by only two functions: the system initialize and the program loop with the system tasks.

```
SYS_Initialize ( NULL );

while ( true ){

      SYS_Tasks ( );

}
```

During the initialization, the PIC oscillator is configured as external with a multiplier. This allows the microcontroller to run at its maximum speed, 80MHz. In addition, the GPIO pins are configured. After that, all the peripherals are initialized using the following libraries.

Libraries	
CAN	Functions to initialize, send and receive CAN messages
USB	Functions to initialize the USB as host and establish connection with the microcontroller
UART	Functions to initialize, send and receive UART messages
File System	Functions to manage files (create, read, write…).
Timer	Functions to control interrupts

Table 3. PIC libraries. Source: Own.

Once every component is initialized, the program enters in the loop and executes the system tasks, which include the interruptions and the control of the state machine.

8.1.4. Data files

One of the main solutions for making the module more versatile is to accept modifications in the data received. That is why a new data organization is implemented in this prototype. The keys are runnable data and a good structure.

```
struct info{                  //Data information for each variable

    char name[10];            //Variable name

    int lenght;               //Number of bytes

    int gain;                 //Gain

    int offset;               //Offset

    int value;                //Variable value

};

struct CAN_ID{                //CAN information

    int ID;                   //ID

    struct info data[8];      //Contained variables

};
```

Creating arrays of CAN_ID structures (see example below) allow any user to easy modify variables and Can messages in the function `set_data()`.

```
struct CAN_ID IDs[ID_number];                 //ID list

void set_data(){

   IDs[0].ID = 0x201;                         //First ID

      strcpy(IDs[0].data[0].name,"TIME");     //Variable 1 name
      IDs[0].data[0].lenght = 1;              //N° of bytes
      IDs[0].data[0].gain = 0;                //Gain
      IDs[0].data[0].offset = 0;              //Offset

      strcpy(IDs[0].data[1].name,"FAIL");     //Variable 2 name
      IDs[0].data[1].lenght = 1;              // N° of bytes
      IDs[0].data[1].gain = 0;                // Gain
      IDs[0].data[1].offset = 0;              // Offset
```

8.1.5. State machine

The program is structured as a state machine, being different functions executed in each state:

OPEN HOST LAYER: starts the USB module as host

WAIT FORT HOST ENABLE: waits for the USB host configuration.

WAIT FOR DEVICE ATTACH : waits for an USB device plug

DEVICE CONNECTED: pass to MOUNT DISK if a device is plugged

MOUNT DISK: starts the connection with the memory of the USB device

WAIT DSPACE: waits for the CAN start message

OPEN FILE: creates a new file in the USB device

TRANSMIT DSPACE: transmit de response to Dspace through CAN

WAIT VARIABLES: start the variables

ASSIGN VARIABLES: receive the messages and assign value to each variable

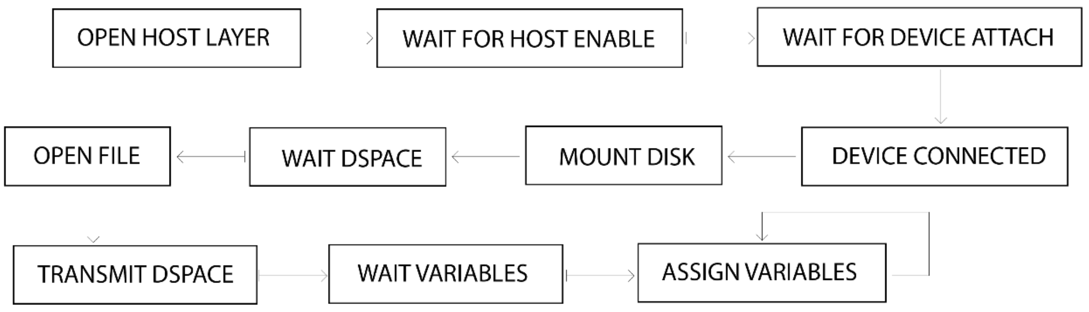


Figure 13. State machine. Source: Own.

If any of these states do not succeed, the program jumps to an error state to reset the program.

8.1.6. Interruption

To make the ECU more versatile and stop relying on the dSPACE ECU, the software uses an interrupt to run the variables and send it every ten milliseconds. By using interrupts, the acquisition of the data can be done at different frequencies avoiding the saturation of the CAN network.

8.2. WiFly Software

In our application, the module has to receive bytes through UART, create a WLAN and send the information via TCP to client devices.

The WiFly module has two modes of operation: Data mode and Command mode. In data mode, which is usually used, the module can accept incoming connections or initiate outgoing connections. To configure the module, you must put it in *Command* mode.

The easiest way to configure the module is via Wi-Fi with the app “WiFly apps”. To access command mode, you must send to the module the characters $$$ together.

The commands used to configure the module are listed in the following table.

Command	*Description*
Set uart baud 115200	Sets the uart baudrate to 115200
Set uart flow 0x00	Sets the flow control mode and parity to none.
Set ip dhcp 4	Enables DHCP mode
Set wlan join 7	Create a soft AP network using the stored SSID, IP address, netmask, channel, etc.
Set wlan ext_antenna 1	Sets the antenna to external
Set wlan ssid CAT09e	Sets the SSID

8.3. App Software

8.3.1. Java Language

Java is a general purpose object oriented computer programming language. Java code can run on all platforms that support Java without the need for recompilation. The syntax of Java is largely influenced by C++. It was built almost exclusively as an object-oriented language. All code is written inside classes, and every data item is an object, with the exception of the primitive data types. Java does not compile to native processor code but rather it relies on a “virtual machine” which understands an intermediate format called Java bytecode. Each platform that runs Java needs a virtual machine (VM) implementation.

The official android programming language, used in the development tool *Android Studio*, is Java complemented with XML for the interface design.

Figure 14. Android Studio Logo. Source: Google

There are some alternatives to Android Studio for android developing such as *Corona*, *Basic4Android* or *Phonegap* that do not use Java.

Corona is a high level SDK built on the LUA programming language. LUA is much simpler to learn than Java and the SDK takes away a lot of the complex syntax in developing Android apps.

Basic4Android is a rapid application development tool for Android applications. It uses a language similar to Visual Basic and has a visual designer that makes easy the process of designing the interface. It also has many APIs, allowing to speed up work.

PhoneGap is an open source framework for quickly building mobile apps for different platforms using HTML5, Javascript and CSS. It is useful if you are used to program websites.

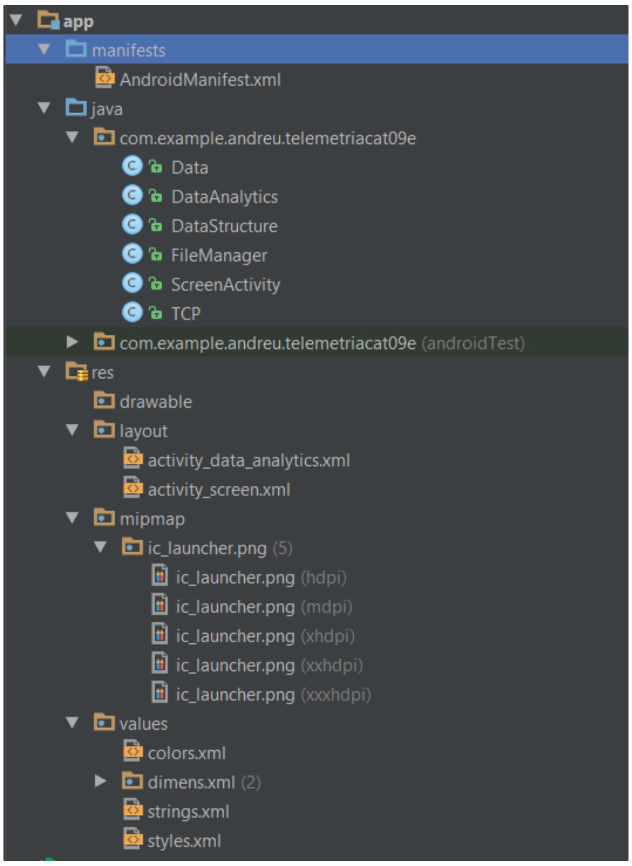
In this project, the *Android Studio* platform has been used to learn the basics of the Android development and Java programming language before using an embedded product for a faster application development.

8.3.2. App structure

An Android project contains everything that defines the Android application, from app source code to build configurations and test code. The SDK tools require that your projects follow a specific structure so it can compile and package your application correctly.

The modules created with the project are: the Android Application Modules, the Test Modules and the Library Modules.

The Android Application Modules include all the source code, resource files, manifest file, etc., of the application. From this module, the .apk file used to install the app into the device will be created. Our application includes one manifest file, six java files, two layout xml files and the other resources.



```
▼ app
  ▼ manifests
      AndroidManifest.xml
  ▼ java
    ▼ com.example.andreu.telemetriacat09e
        Data
        DataAnalytics
        DataStructure
        FileManager
        ScreenActivity
        TCP
    ▶ com.example.andreu.telemetriacat09e (androidTest)
  ▼ res
      drawable
    ▼ layout
        activity_data_analytics.xml
        activity_screen.xml
    ▼ mipmap
      ▼ ic_launcher.png (5)
          ic_launcher.png (hdpi)
          ic_launcher.png (mdpi)
          ic_launcher.png (xhdpi)
          ic_launcher.png (xxhdpi)
          ic_launcher.png (xxxhdpi)
    ▼ values
        colors.xml
      ▶ dimens.xml (2)
        strings.xml
        styles.xml
```

Figure 15. Application Structure. Source: Own

The Test Modules contain the code to test the applications.

The Library Modules contain all the libraries used in the project such as the Android Plot library.

8.3.3. App classes and functionality

As mentioned before, the application is developed with the *Android Studio* software. The app receives TCP packets with the variables and shows them. All the process is done with classes and objects written in Java and it is shown via the XML layout and resource files. The classes of the program are explained below and its significant parts are shown.

- Data Structure: Class that defines the data format type. Contains the characteristics of each variable and functions to interact with it. All the variables of the car will be objects of this class with different characteristics.

The content of each object is private and can only be modified through the setter functions.

Every variable has its name that will be its unique ID. The length represents the number of bytes that the variable has, this is needed when variables are split in the Wi-Fi message. The gain and the offset are applied to the received data to get the final value. Finally, the raw data are the values with no gain and offset.

```
public class DataStructure {
    //Variables
    private String name;
    private int lenght;
    private int gain;
    private int offset;
    private long value;
    private long rawdata;

    //Getters
    public String getName(){..}
    public int getLenght(){..}
    public int getGain() {..}
    public int getOffset() {..}
    public long getValue() {..}
    public long getRawdata() {..}

    //Setters
    public void setName(String name) {..}
    public void setGain(int gain) {..}
    public void setLenght(int lenght) {..}
    public void setOffset(int offset) {..}
    public void setValue(long value) {..}
    public void setRawdata(long rawdata) {..}
           /// this function transforms the data with its
               gain and offsets and sets the value
```

- Data: Contains the array of data and functions to interact with it. The user must modify this file to change the variables.

  Using an arraylist allows data to be runnable. That makes the insertion of data values easy.

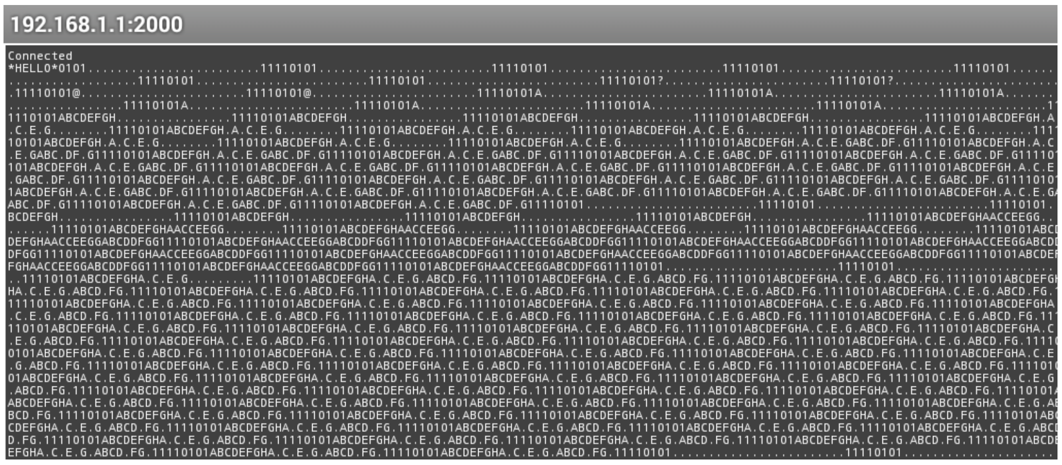
  The `dataInfo.add()` function adds a new DataStructure object with its name, length, gain, offset and raw value to the arraylist.

```
public class Data {
    private   ArrayList<DataStructure>   dataInfo   =   new
ArrayList<>();
    public Data(){
        dataInfo.add(new DataStructure("TIME",4,1,0,0));
        dataInfo.add(new DataStructure("TIMD",1,1,0,0));
        dataInfo.add(new DataStructure("FAIL",1,1,0,0));
        dataInfo.add(new DataStructure("TMAX",1,1,0,0));
        dataInfo.add(new DataStructure("CMAX",1,1,0,0));
        dataInfo.add(new DataStructure("VMN5",1,1,0,0));
    }
    public ArrayList getData(){..}
    public int getDataLenght(){..}
    public Long getSingleValue(String name){ ..}
    public DataStructure getSingleData(String name){ ..}
    public ArrayList getNames(){..}
    public ArrayList getValues(){..}
    public void setRawData(ArrayList<Integer>
rawDataList){ ..}
     //This function call the setRawData() function for
each member of the array taking in account the length of
each variable.


}
```

- File Manager: Contains functions to create, delete and modify files.

```
Public class FileManager {
  public String dir = "";
    public String filename = "";
    public File file;

    public void setDir(String directory){..}
    public void setFileName(string filename){..}
    public void createFile(){..}
    public void rename{..}
    public void savearray(ArrayList data) {..}
           //This function save any array with “,” between
the elements
```

- TCP: This class establishes the connection with the Wifly module and receives TCP messages. It is also the bridge between the previous classes and the Screen Activity class. Figure 16 shows the transferred data via Wi-Fi. This class also interprets this data before assigning it to the variables.



192.168.1.1:2000

```
Connected
*HELLO*0101.......................11110101.......................11110101.......................11110101.......................11110101.......................11110101......
.......................11110101.......................11110101.......................11110101?.......................11110101?......................
.11110101@.......................11110101@.......................11110101A.......................11110101A.......................11110101A.......
.....................11110101A.......................11110101A.......................11110101A.......................11110101A.......................11
1110101ABCDEFGH...............11110101ABCDEFGH...............11110101ABCDEFGH...............11110101ABCDEFGH...............11110101ABCDEFGH.A
.C.E.G.......11110101ABCDEFGH.A.C.E.G.......11110101ABCDEFGH.A.C.E.G.......11110101ABCDEFGH.A.C.E.G.......11110101ABCDEFGH.A.C.E.G.......1111
10101ABCDEFGH.A.C.E.G.......11110101ABCDEFGH.A.C.E.G.......11110101ABCDEFGH.A.C.E.G.......11110101ABCDEFGH.A.C.E.GABC.DF.G11110101ABCDEFGH.A.C
.E.GABC.DF.G11110101ABCDEFGH.A.C.E.GABC.DF.G11110101ABCDEFGH.A.C.E.GABC.DF.G11110101ABCDEFGH.A.C.E.GABC.DF.G11110101ABCDEFGH.A.C.E.GABC.DF.G111101
101ABCDEFGH.A.C.E.GABC.DF.G11110101ABCDEFGH.A.C.E.GABC.DF.G11110101ABCDEFGH.A.C.E.GABC.DF.G11110101ABCDEFGH.A.C.E.GABC.DF.G11110101ABCDEFGH.A.C.E
.GABC.DF.G11110101ABCDEFGH.A.C.E.GABC.DF.G11110101ABCDEFGH.A.C.E.GABC.DF.G11110101ABCDEFGH.A.C.E.GABC.DF.G11110101ABCDEFGH.A.C.E.GABC.DF.G1111010
1ABCDEFGH.A.C.E.GABC.DF.G11110101ABCDEFGH.A.C.E.GABC.DF.G11110101ABCDEFGH.A.C.E.GABC.DF.G11110101ABCDEFGH.A.C.E.GABC.DF.G11110101ABCDEFGH.A.C.E.GA
ABC.DF.G11110101ABCDEFGH.A.C.E.GABC.DF.G11110101ABCDEFGH.A.C.E.GABC.DF.G11110101.......................11110101.......................11110101.
BCDEFGH...............11110101ABCDEFGH...............11110101ABCDEFGH...............11110101ABCDEFGH...............11110101ABCDEFGHAACCEEGG..
......11110101ABCDEFGHAACCEEGG.......11110101ABCDEFGHAACCEEGG.......11110101ABCDEFGHAACCEEGG.......11110101ABCDEFGHAACCEEGG.......11110101ABCD
DEFGHAACCEEGGABCDDFGG11110101ABCDEFGHAACCEEGGABCDDFGG11110101ABCDEFGHAACCEEGGABCDDFGG11110101ABCDEFGHAACCEEGGABCDDFGG11110101ABCDEFGHAACCEEGGABCDD
DFGG11110101ABCDEFGHAACCEEGGABCDDFGG11110101ABCDEFGHAACCEEGGABCDDFGG11110101ABCDEFGHAACCEEGGABCDDFGG11110101ABCDEFGHAACCEEGGABCDDFGG11110101ABCDEF
FGHAACCEEGGABCDDFGG11110101ABCDEFGHAACCEEGGABCDDFGG11110101ABCDEFGHAACCEEGGABCDDFGG11110101.......................11110101.......................
..11110101ABCDEFGHA.C.E.G.......11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGH
HA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.1
11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA
.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.111
110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C
.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.1111
0101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E
.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.111101
01ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G
.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101A
ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.AB
BCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABC
CDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD
D.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDE
EFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101ABCDEFGHA.C.E.G.ABCD.FG.11110101.......................11110101.......................
```

Figure 16. Transferred data. Source: Own

The decoding process starts by searching the four flags of the start message, the numbers 0,1,0,1 and assigning the rawvalue to each variable depending on its length.

```
public class TCP extends AsyncTask{
//Variables

    public Data data = new Data();          //Data
    public FileManager fileManager = new FileManager();
    private String ip = "192.168.1.1";      //Host IP
    private int port = 2000;                // Host Port
    private Socket s;                       //Socket
    private boolean hostConnection = false; //Connection
Status
    private int savePeriod = 10;   //Actualization period
    private ArrayList<Integer> dataArray = new
ArrayList<>(); ;  //Data from TCP
    private Timer saveTimer = new Timer(true);

    // Background process
    @Override
    protected Object doInBackground(Object[] params) {

    //Getters
    public String getIp(){
    public int getPort(){
    public ArrayList<Integer> getTcpData(){
    public boolean isConnected(){

    //Setters
    private void setSavePeriod(int period){
    public void setIp(String ip){
    public void setPort(int port){

    //Class functions
    public void start(){
    public void stop(){
    public void save(final boolean yes){

}
```

- Screen Activity: Creates the Views of the variables and updates them.

```
public class ScreenActivity extends AppCompatActivity {
    TCP tcp = new TCP();

    protected void onCreate(Bundle savedInstanceState) {

        //Button definition
        final Button rec = (Button)
findViewById(R.id.button_rec);
        final Button stop = (Button)
findViewById(R.id.button_stop);
        Button dataAnalytics = (Button)
findViewById(R.id.button_data_analytics);

        //Button actions
         rec.setOnClickListener(..)
           //Starts saving data

         stop.setOnClickListener(..)
           //Stops saving data

         dataAnalytics. setOnClickListener(..)
           //Starts data analytics activity

        tcp.start();
           //If the device is connected to a wifi network
           starts the connection.

        Timer screenRefresh = new Timer(true);
           //Update the screen

        ArrayList<Integer> values = tcp.data.getValues();
            screenRefresh.schedule(new TimerTask() {

                                textView2.setText(..);
            },0,200);
```

- Data Analytics: This class, implemented with the free library "Android Plot" [11] (androidplot.com), makes X-Y graphics of the selected variables from an archive inside the disk.

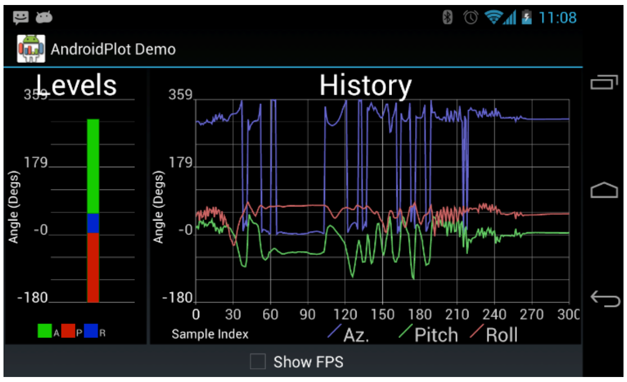


Figure 17. Android Plot demo. Source: Android Plot Website.

9. Assembly, test and validations

This chapter is divided into two sections: the assembly and validation of the hardware and the software tests.

9.1. Hardware – PCB assembly

9.1.1. Power

The first step is soldering the power modules (DC/DC converters) in the PCB. Once they are soldered, a 12V power supply is connected and they are tested to check the correct operation in case of short circuit or voltage rise.

A power indicator led is soldered next to the power modules to verify the proper operation.

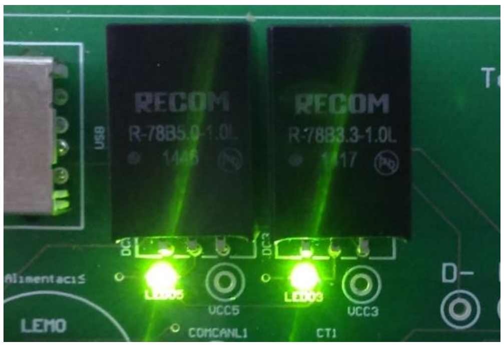


Figure 18. Power modules. *Source: Own.*

9.1.2. Microcontroller

Soldering the microcontroller is a difficult task. For this reason, Flux and a 0,6mm soldering tip has been used to facilitate the work. To test the PIC32, it is connected to the PC through a PICKIT programmer and it is programed with and old program to see if the program runs on it.

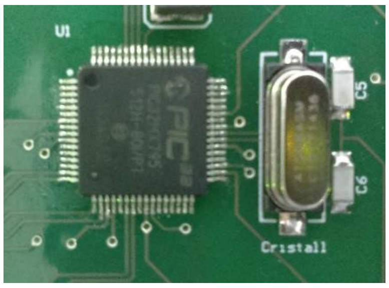


Figure 19. PIC and 8MHz crystal. Source: Own.

When it works as it is supposed to, all the peripherals such as the clock source or the CAN transceivers are soldered and tested one by one.

9.1.3. WiFly module

The WiFly module manufacturer gives few recommendations to solder the module:

- The solder temperature must be below 220ºC
- Don’t clean with water
- Don’t use No Clean Flux

When the module is connected, its functioning can be checked by the notification LEDs according to the datasheet [6].

9.1.4. Final assembly

The following image shows the final assembly of the ECU which will be inside a Hammond Manufacturing case.

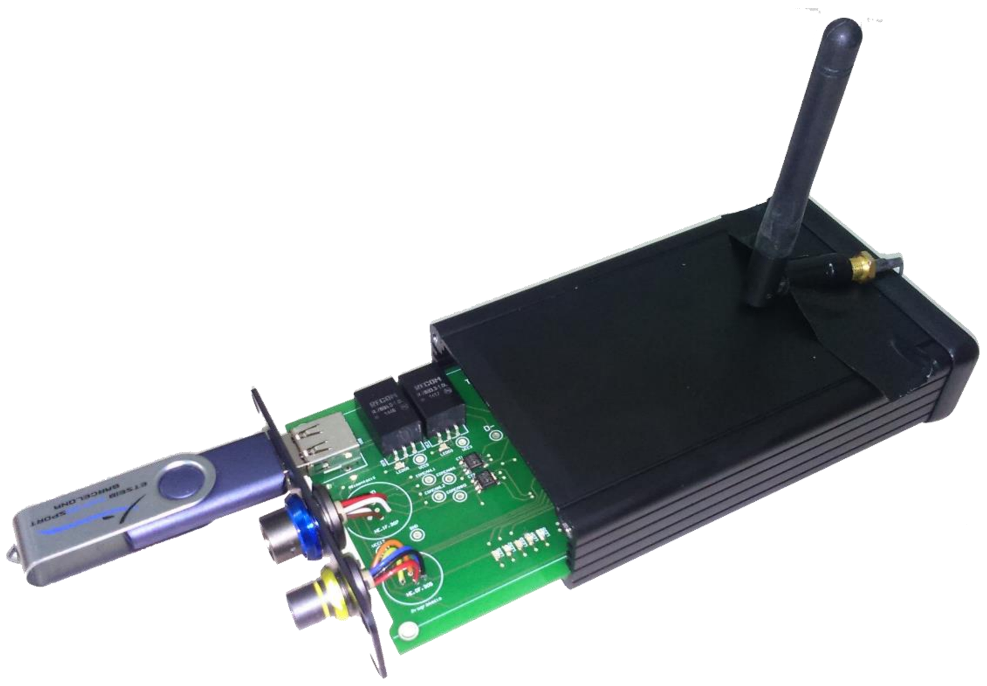


Figure 20. Final assembly. Source: Own.

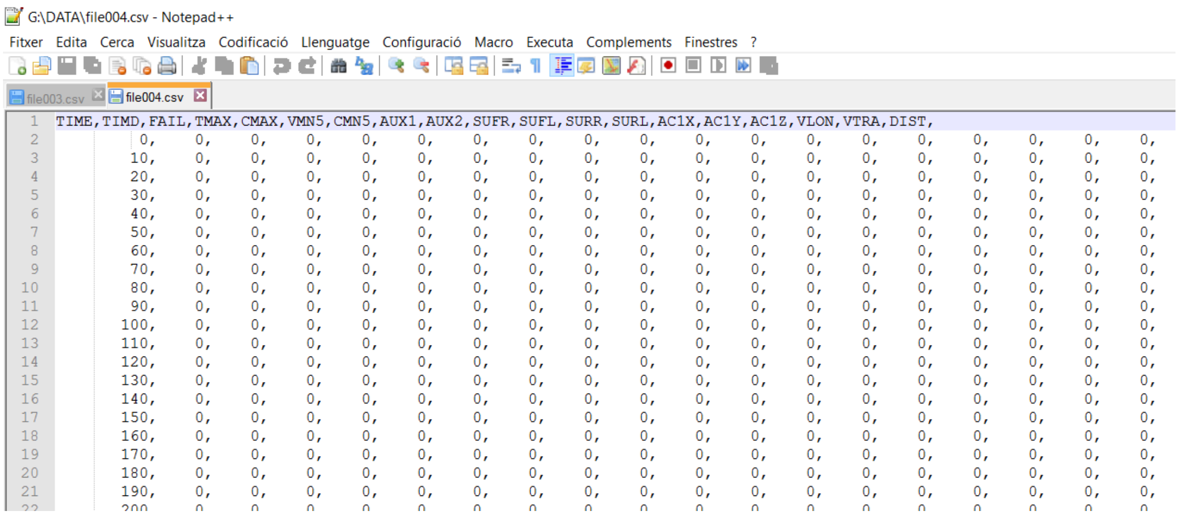
9.2. Software

9.2.1. PCB software

As this project is an evolution of previous projects, the software of the microcontroller is based on CAT08e's software. All the modifications made on it have been tested in the base program to check the compatibility with it.

One of the advantages of the new developed program, the capacity to control the data saving time, has been tested using CAN messages and observing the precision of the frequency in the PC using a CAN interface from Kvaser.

In Figure 21 the output file in the USB can be seen, a csv file that can be imported to the data analysis software GEMS. Every file starts with the name of the variables in the first line starting with TIME and all the other lines are the rest of the data.


G:\DATA\file004.csv - Notepad++
Fitxer Edita Cerca Visualitza Codificació Llenguatge Configuració Macro Executa Complements Finestres ?
file003.csv
file004.csv

```
TIME,TIMD,FAIL,TMAX,CMAX,VMN5,CMN5,AUX1,AUX2,SUFR,SUFL,SURR,SURL,AC1X,AC1Y,AC1Z,VLON,VTRA,DIST,
        0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
       10,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
       20,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
       30,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
       40,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
       50,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
       60,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
       70,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
       80,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
       90,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
      100,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
      110,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
      120,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
      130,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
      140,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
      150,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
      160,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
      170,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
      180,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
      190,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
      200,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
```

Figure 21. Output file. Source: Own

9.2.2. Android APP software

To assembly the final app, several test applications have been made to simulate single parts of the program. Some of these apps are listed below:

- Intent app: Starts an activity by clicking a button
- TCP app: Establishes connection with a host, receive messages and print it
- Timer app: Makes an action at a specified frequency
- Asynctask app: Runs background processes

- File managing app: Reads and Writes to files
- Graphics app: Prints X-Y graphics

The final app joins these small apps in a single one. It has one *rec/stop* button and the *data analytics* button. Once it is opened and the device is connected to CAT09e WLAN, it starts receiving all the data (see Fig. 22).

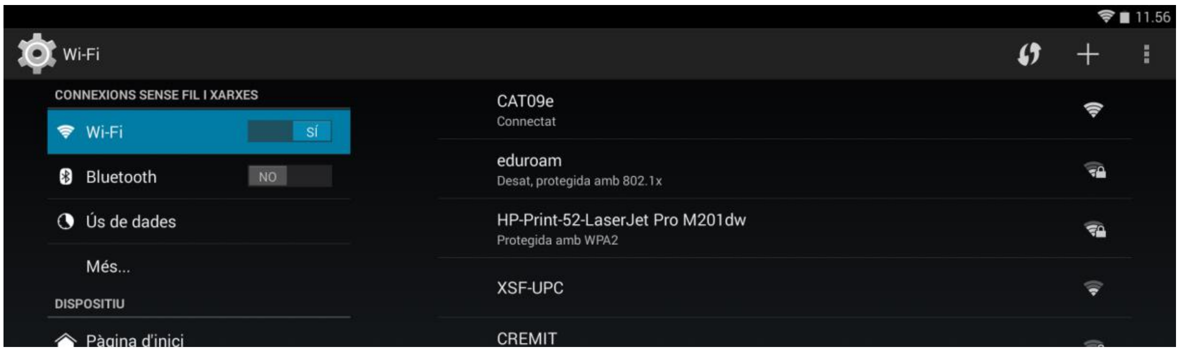
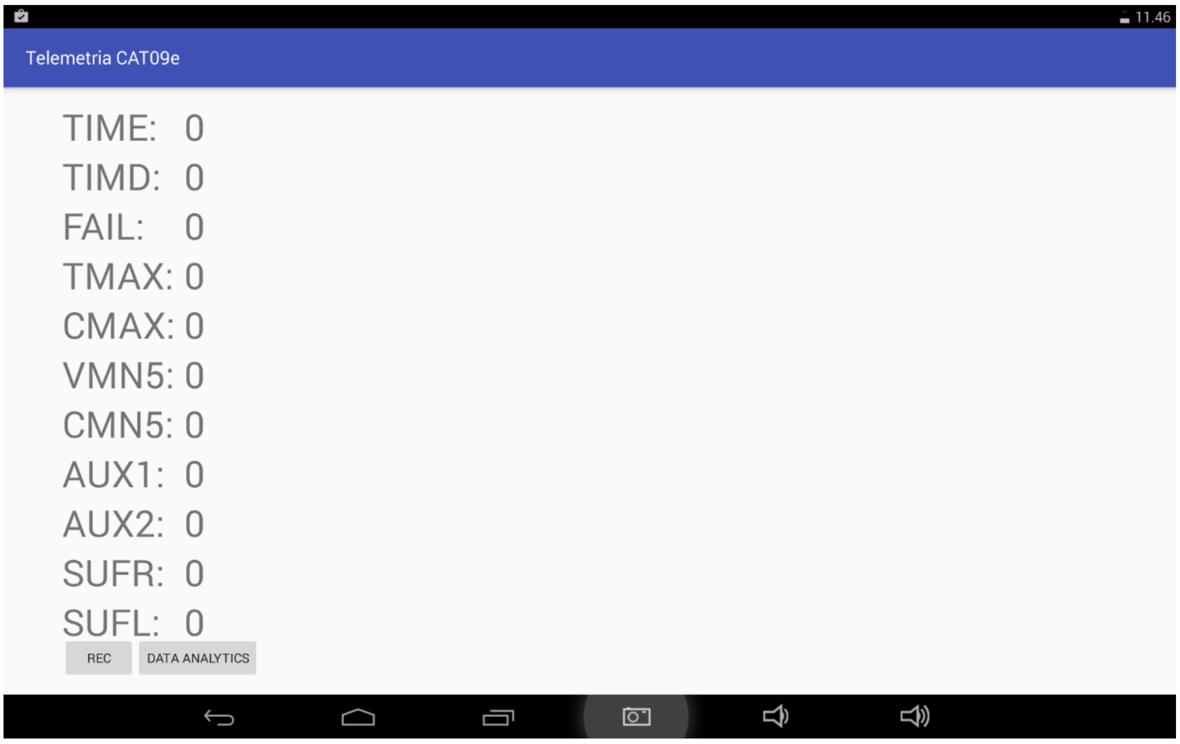


Figure 22. Wi-Fi configuration. Source: Own.


Telemetria CAT09e

TIME: 0
TIMD: 0
FAIL: 0
TMAX: 0
CMAX: 0
VMN5: 0
CMN5: 0
AUX1: 0
AUX2: 0
SUFR: 0
SUFL: 0

Figure 23. Main screen. Source: Own.

To record data the user has to press the button *rec* and when the stop button is pressed a dialog box appears to set the filename. If the dialog is canceled, the run is deleted.

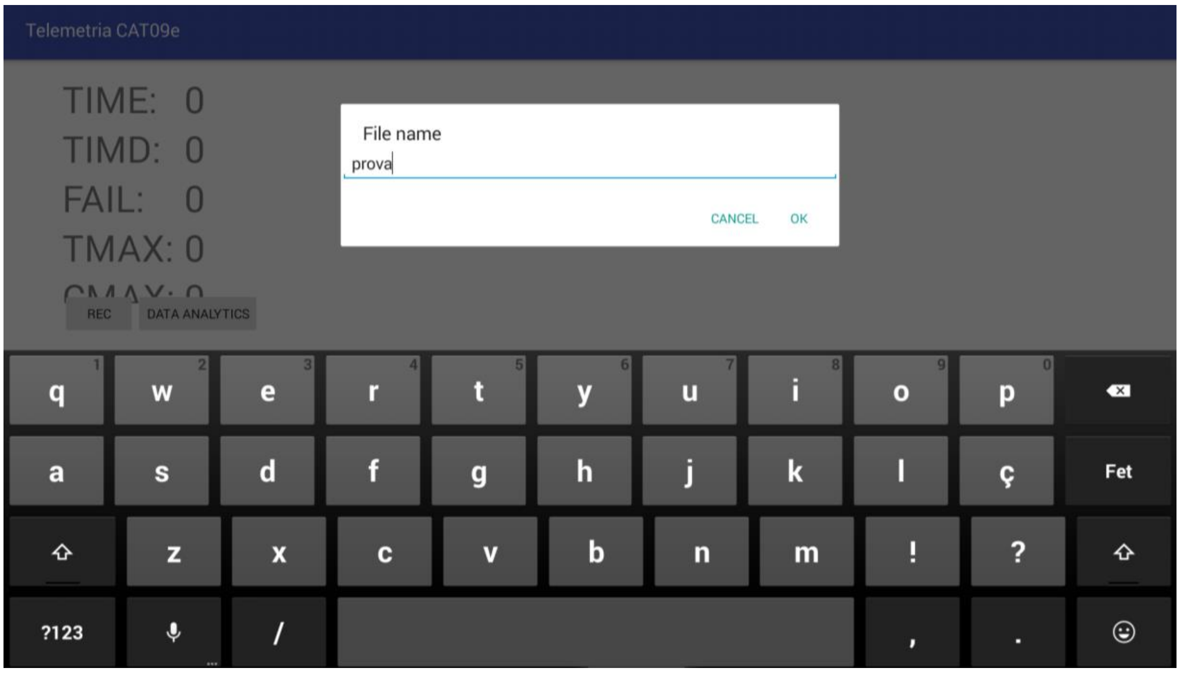


Figure 24. Dialog box. Source: Own.

The Data analytics screen shows the files stored in the system, the variables in the selected file and the graphic of the selected variables.

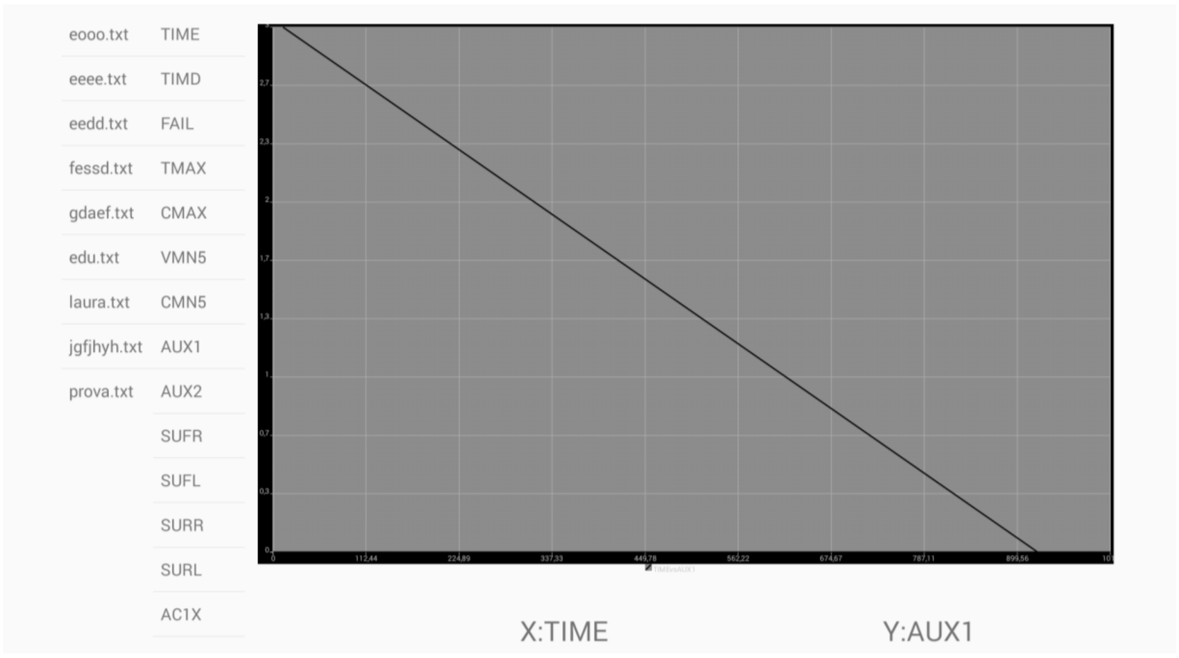


Figure 25. Data Analytics screen. Source: Own.

10. Budget

The following tables detail the costs of the project, including material cost, equipment cost, software cost and personal cost.

10.1. Material Cost

The material cost table includes all the components of the ECU.

	PART	COST
MATERIAL COST	Res 10	0,09 €
	Res 100	0,36 €
	Res 120	0,18 €
	Res 220	0,09 €
	Res 260	0,82 €
	Res 470	0,18 €
	Res 1000	0,18 €
	Res 4700	0,18 €
	Res 10000	0,18 €
	Res 100000	0,18 €
	Cond 20pF	0,27 €
	Cond 560pF	0,36 €
	Cond 100nF	1,99 €
	Cond 1uF	1,17 €
	Cond 10uF	0,26 €
	Cond 47uF	1,14 €
	CEFC 10uF 1ohm	7,63 €
	XTAL 8MHz	0,33 €
	CAN Transceiver	5,20 €
	DC5	8,44 €
	DC3.3	8,44 €
	Led Green	1,11 €
	Led Red	0,24 €
	Led Yellow	0,29 €
	Led Blue	0,67 €
	RN-131C/RM Wifi Module	33,42 €
	USB	2,49 €
	Zener 3.3V	0,18 €

	Box Hammond 1455J1201	18,81 €
	LEMO 0F305	105,11 €
	LEMO 0F307	114,85 €
	PIC 32	7,94 €
	USB	6,99 €
	PCB	290,00 €
	Subtotal	**619,78 €**

Table 4. Material Cost.

10.2. Equipment cost

EQUIPMENT COST	Personal Computer	700,00 €
	Power Supply	948,00 €
	Oscilloscope	798,00 €
	Kvaser Leaf light 2	326,28 €
	Multimeter	50,00 €
	Pickit 3	42,50 €
	Android Device	200,00 €
	Subtotal	**3.064,78 €**

Table 5. Equipment Cost.

10.3. Software cost

SOFTWARE COST	Altium designer	6.795,00 €
	MPLAB X	0,00 €
	MPLAB Harmony	0,00 €
	Kvaser CanKing	0,00 €
	Android Studio	0,00 €
	Matlab	2.000,00 €
	Subtotal	**8.795,00 €**

Table 6. Software Cost.

10.4. Personal cost

	PART	HOURS	COST
PERSONAL COST	Project Study	152	7.600,00 €
	Hardware design	144	7.200,00 €
	PCB Assembly	16	800,00 €
	Software Design	68	3.400,00 €
	Validations	48	2.400,00 €
	Subtotal	**428**	**21.400,00 €**

Table 7. Personal Cost

10.5. Total cost

As it can be seen in Table 8, the total cost of the project is around 33.800€. The 63% of the cost is due to human resources and the other 34% corresponds to the equipment and software licenses. This ECU can be made because the project is voluntary and all the equipment and software is provided by sponsors.

TOTAL COST	Material Cost	619,78 €
	Equipment Cost	3.064,78 €
	Software Cost	8.795,00 €
	Personal Cost	21.400,00 €
	Total	**33.879,56 €**

Table 8. Total Cost

ETSEIB

11. Environmental impact

The environmental impact analysis of this project is focused on two points: the electronic components used in the PCB and the electromagnetic waves produced by the Wi-Fi module.

The RoHS directive (Restriction of Hazardous Substances) regulates the use of some materials in electric and electronic components. These materials, such as Mercury or Cadmium, represent an environmental hazard, therefore they must be recycled in specific plants. Since 2006, the members of the RoHS directive forced companies to stop using these substances.

All the components of the PCB and the Android device are RoHS compliant.

The other point of the environmental impact analysis, as mentioned before, is the electromagnetic effect of the Wi-Fi signal to the human body. Even though there are not clear evidences of this effect, the module manufacturer establishes that the module can be used at a distance of at least 20cm from the body without consequences for human health.

Conclusions

The main objective of the project is the implementation of an ECU according to the needs of the Formula Student car CAT09e and an Android application to read car data. It started from two ECUs used in previous cars that did not work as expected. This project wanted to optimize both software and hardware.

These goals have been achieved, as a fully operational ECU has been assembled and tested. The code, which is much more understandable, more structured and easy to modify, has also been optimized.

The application, despite having possible improvements, is able to read data in real time, save it and show X-Y graphics on a tablet display.

Using both solutions together optimizes testing time and allows a faster decision making to get the best performance of the car.

As a further improvement, a bidirectional communication between the Android application and the car should be implemented. This fact was already stated in the CAT07e Telemetry project but never implemented because was not a priority. In addition, the application could have more options and displays to show different type of data.

Acknowledgements

In the first place, I would like to thank my Formula Student Colleagues. The continuous work and discussion over hundreds of hours has been an invaluable source of knowledge and expertise in group projects.

I would also like to thank my project supervisor, Manuel Moreno, for his significant technical support as well as for his important contribution to the team’s electronic department.

Finally, I also want to thank my family for the support they give me every day in my education.

ETSEIB

Bibliography

Bibliographic references

[1] GASCÓN DOMINGO, D. *Telemetria basada en WIFI per a l'equip ETSEIB Motorsport Barcelona de la Formula Student*. Barcelona, ETSEIB, 2014.

[2] CABUTÍ BORRELL, P. *Disseny i millora del sistema d'enregistrament de dades per a l'equip ETSEIB Motorsport de la Formula Student*. Barcelona, ETSEIB, 2015.

[3] KVASER. *Kvaser Leaf Light 2 User's Guide*. 2015. [https://www.kvaser.com/products/kvaser-leaf-light-hs-v2/, December 2015]

[4] MICROCHIP TECHNOLOGY. *PIC32MX795F512H Datasheet*. 2015. [http://ww1.microchip.com/downloads/en/DeviceDoc/61156H.pdf, December de 2015]

[5] MICROCHIP TECHNOLOGY. *PIC32 Family Reference Manual*. 2015. [http:// www.microchip.com, December de 2015]

[6] MICROCHIP TECHNOLOGY. *RN131 Datasheet*. 2014. [http://ww1.microchip.com/downloads/en/DeviceDoc/rn-131-ds-v3.2r.pdf, December 2015]

[7] MICROCHIP TECHNOLOGY. *WiFly Module Command Reference User's Guide*. 2014. [http://ww1.microchip.com/downloads/en/DeviceDoc/rn-131-ds-v3.2r.pdf, December 2015]

[8] MICROCHIP TECHNOLOGY. MPLAB Harmony v1.03.01 HELP. 2015 [http://www.microchip.com/pagehandler/en_us/devtools/mplabharmony/home.html]

[9] NEIL SMITH. Android Studio Development Essentials. 2015

[10] GOOGLE. *Android developers*. 2015 [http://developer.android.com/index.html, December 2015]

[11] Android Plot. 2015 [http://androidplot.com, December 2015]

Complementary bibliography

FORMULA SAE. *2016 Formula SAE® Rules*.
[http://students.sae.org/cds/formulaseries/rules/, January 2015]

FORMULA STUDENT GERMANY. *Formula Student Electric Rules 2015*. 2015.
[http://www.formulastudent.de/fse/2015/rules/, January 2015]

ETSEIB