INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Automotive data acquisition system - FST

David Rua Copeto
nº 53598

Dissertation for obtaining the Master's Degree in

Electrical and Computer Engineering

Jury

President:	Prof. Marcelino Santos
Advisor:	Prof. Francisco Alegria
Co-Advisor:	Prof. Moisés Piedade
Specialist:	Prof. Nuno Roma

October 2009

Resumo

Esta tese aborda o design, a implementação e a validação de um sistema de telemetria para um protótipo Formula Student, tendo em mente uma rede de sensores suportada num barramento CAN, existente neste. Para o conseguir, o sistema proposto é dividido em dois blocos: uma estação móvel e uma estação base. A primeira, é colocada no veículo e ligada aos seus sensores através do barramento CAN. Esta estação móvel tem a função de gravar localmente os dados gerados pela actividade no barramento e também de transferir sem fios, estes dados, para a estação base fora da pista. A segunda, tem a função de pegar nos dados recebidos através do canal sem-fios e de os apresentar ao utilizador de uma forma atraente e compreensível. Para além do funcionamento "online", a estação base também permite a apresentação de dados relativos a sessões anteriores para análise.

Dado o tipo de veículo (e competição) a que este trabalho se aplica, existem algumas exigências tanto em termos de capacidade do sistema, como de gestão do projecto. Por um lado, o sistema deve ser capaz de resistir a ambientes adversos, nomeadamente vibração, calor, líquidos e interferência electromagnética, e por outro deve ser leve, barato e fácil de utilizar.

O sistema de telemetria desenvolvido foi utilizado com sucesso numa pista de treinos, sendo capaz de gravar com fiabilidade os dados provenientes do barramento CAN com um número, frequência de amostragem e tipo variável de sensores. A performance do canal de rádio entre estações foi também obtida, tanto num ambiente controlado como real, de maneira a perceber quais os limites do sistema. Os resultados mostram uma taxa de transferência de dados sem-fios de 4.28 kB/s para uma localização típica do veículo em pista.

Palavras chave: Telemetria, barramento CAN, Zigbee, Formula Student, gravação de dados em ambiente automóvel, interface gráfica do utilizador.

Abstract

This thesis addresses the design, implementation and validation of a telemetry system for a Formula Student prototype vehicle, having in mind an existing CAN-bus network of sensors in it. To achieve this, the proposed system is divided in two blocks: a mobile station and a base station. The first, is placed on the vehicle and is connected to its sensors through a CAN-bus. This mobile station has the function of recording locally the data generated by the bus activity and also of transferring this data wirelessly to the base station located off-track. The second has the function of picking up the data sent through the wireless link and presenting it in an attractive and comprehensible way to the user. Besides the "online" operation, the base station is also capable of presenting data from previous logging sessions for analysis.

Given the type of vehicle (and competition) this work is applied to, certain requirements are set in terms of both system capabilities and project management. In one hand the system must be able to sustain a harsh environment, namely vibration, heat, liquids and electromagnetic interference, and on the other hand it must be lightweight, cheap and user-friendly.

The developed telemetry system was successfully deployed on a practice track, being able to reliably record CAN-bus data of a variable number, sample rate and type of sensors. The performance of the radio link between stations was also evaluated both in controlled and real environments in order to understand the limits of the system. The results show a maximum wireless data rate of 4.28 kB/s for a typical vehicle position on track.

Keywords: Telemetry, CAN-bus, Zigbee, Formula Student, automotive data logger, graphical user interface.

Contents

List of Tables

List of Figures

List of Abbreviations

ACPR	Adjacent Channel Power Ratio
ASCII	American Standard Code for Information Interchange
BCD	Binary-coded Decimal
CAN	Controller Area Network
CF	Compact Flash
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CRC	Cyclic redundancy check
CSMA/CD	Carrier Sense Multiple Access/Collision Detection
EEPROM	Electrically-Erasable Programmable Read-Only Memory
EMI	Electromagnetic Interference
FAT	File Allocation Table
FIFO	First In, First Out
FSPL	Free Space Path Loss
FTDI	Future Technology Devices International
GPS	Global Positioning System
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
I²C	Inter-Integrated Circuit
IC	Integrated Circuit
IMechE	Institution of Mechanical Engineers
ISM	Industrial, Scientific and Medic
ISO	International Standards Organization
IST	Instituto Superior Técnico

KIP	Kartódromo Internacional de Palmela
LED	Light-Emitting Diode
MCU	Microcontroller Unit
MMC	Multimedia Card
OS	Operating System
PC	Personal Computer
PCB	Printed Circuit Board
PDIP	Plastic Dual-in-line Package
PIC	Peripheral Interface Controller
PLL	Phase-Locked Loop
RF	Radio Frequency
RPM	Revolutions Per Minute
RSS	Received Signal Strength
RTC	Real-Time Clock
SAE	Society of Automotive Engineers
SD	Secure Digital
SMA	SubMiniature version A
SMD	Surface Mount Device
SNR	Signal to Noise Ratio
TCP	Transmission Control Protocol
TCXO	Temperature-compensated Crystal Oscillator
TTL	Transistor-Transistor Logic
UHF	Ultra High Frequency
USART	Universal Synchronous Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VDI	Verein Deutscher Ingenieure

Chapter 1

Introduction

A great deal of time, effort and expense goes into the construction of a prototype. It is usually the product of the work of a large group of people and represents the tangible result of the breakthroughs accomplished after many hours of research. It is of major importance then, to have the best enlightenment possible on it's performance and maintenance conditions. With this purpose in mind, it's easy to understand why nowadays sensor networks and their data gathering systems are a mandatory part of any fairly complex machine. Besides being vital in terms of behavior evaluation, this additional investment is the base on which great cost and time savings can be generated.

Commonly referred to as data loggers, the devices that sample real world phenomenons and store data that can be manipulated by a computer, range from commercial airliner's airborne recorders to portable heart rate monitors. Data acquisition and logging systems can have many different purposes and capabilities, but the main building blocks remain the same:

Figure 1.1: General outline of a digital data logging system.

An important field where data loggers have been used extensively is the automotive area. Specially in competition, data logging plays an important role, allowing a much easier tuning of vehicles, monitoring of critical parts (such as the engine) and as a useful tool for assessing driver performance. Current solutions found on the market offer already a great number of options in terms of number of channels, acquisition rates and bus interfaces, suitable for different kinds of competitions and vehicles. The most common features found in Commercial off-the-shelf (COTS) data loggers aimed at competion vehicles are the ability to record wheel speed, temperature, pressure, engine revolutions per minute (RPM) and pedal travel. More complete systems may also include suspension travel or acceleration in different axis.

Working together with data logging with the objective of quicker access to data, telemetry is nowa-

days an increasingly wanted feature. By adding wireless transmission hardware to a data logger, gathered data can be analyzed right after being generated, and even in real-time, allowing actions to be taken much quicker. In certain applications, where prototypes may be destroyed (e.g. space exploration systems), the usage of telemetry is the only way to recover performance data. In competition vehicles, telemetry is typically used to monitor operational parameters and alert for dangerous or failure situations; in case of two-way communication it may also be used to adjust some parameters.

1.1 Project scope and objective

This work is primarily aimed at a particular prototype vehicle (figure 1.2), built at Instituto Superior Técnico (IST), to participate in the Formula Student competitions[1]. These competions, organized by the Society of Automotive Engineers (SAE) and other engineers societies (IMechE[2], VDI[3], others), are held every year in different countries, not as part of a world championship, but as individual events, each one with its own character and challenges. In these events, teams of students compete with their prototypes in many different ways, from the dynamic performance of the vehicles on track to the way they manage their team. More information about Formula Student and the team from IST - Projecto FST, can be found in appendix A.

Figure 1.2: The prototype FST03 during a test run.

Developing the system for a Formula Student prototype, sets certain requirements in terms of both system capabilities and project management. In one hand the system must be able to sustain a harsh environment, namely vibration, heat, liquids and electromagnetic interference, and on the other hand it must be lightweight, cheap and user-friendly.

[1] The competions held worldwide are here generally called Formula Student. The correct name should be Formula SAE® Series

[2] Institution of Mechanical Engineers - The British engineering society

[3] Verein Deutscher Ingenieure - The Association of German Engineers

The project presented here has the objective of designing and building a fully working CAN-bus data logger with telemetry capabilities for a Formula student prototype vehicle. The system should be able to connect with an existing CAN-bus network of sensors and assure the recording of all the data put on the CAN-bus. Besides this, it should be able to transmit live sensor data through a radio link while the vehicle is running on the track to a location off-track, and making it available for the user in an atractive way. Finally, the built system should be validated in a real environment, that is, working in the prototype vehicle its meant for (figure 1.2).

In the following section the current comercial solutions for the logging of CAN-bus data and competion vehicle telemetry are presented and discussed.

1.2 State of the art

The current range of products found on the market with the purpose of logging sensor data in a vehicle usually offer the data acquisition module separated from the radio module. Recording directly from sensor inputs to a flash memory, the *DL1* from *Race Technology* [21] is a standard in the data loggers range. This system goes a bit further by integrating accelerometers in the system as well as a high acuracy GPS receiver. This kind of stand-alone data loggers can be found in major manufacturers of race vehicle's instrumentation and have the advantage of a pretty straight forward implementation. They are also built having in mind a strong integration with a driver display console, thus in order to make the recorded data available, it has to be downloaded from the memory card.

To be able to implement telemetry from the vehicle, radio modules are found separated from the data acquisition systems. *Coswort* [2] (former *Pi Technology*) offers a radio system, for data transmission from vehicle to pits, working in the 458 to 468 MHz band at 1 Watt and with a data rate of aproximately 16.5 kbps. A low throuput rate if the type of data loggers this system is supposed to work with are considered (the *Pi Sigma Elite Junior* data logger has 16 analog and 8 digital inputs). The functioning of these systems rely on the frequency at which data is sent through the radio link or even on the detection of proximity between radio stations by signal strength, in order to decide when to transmit data.

Manufacturers start now to increasingly offer telemetry links relying on WLAN technology (WiFi 802.11b/g working on the 2.4 GHz frequency band), because it offers much higher data rates (11 or 54 Mbps) - *Stack*'s [17] *MFR-W* and *Cosworth*'s *Pi Ethernet Hub wireless version*. Although the throughput is an undeniable advantage, these systems cannot match the range of lower frequency bands' technologies that are necessary in certain situations (the full coverage of a Formula 1 race track presents a challenge due to the amount of obstructions).

For the logging of CAN-bus messages various solutions for USB interface are found in the market, *Kvaser*'s [12] *Leaf Light* or *LAWICEL*'s [13] *CANUSB*, and some manufacturers make available stand-alone CAN-bus recorders that have specially in mind the industrial use of the CAN-bus, like *Kvaser*'s line of CAN-bus data loggers (*Memorator* line).

Also industrial but on a different segment, *Telemotive*'s [1] *Blue piraT*, is aimed at automotive testing by logging most buses used in automotive industry - MOST, CAN, LIN and FlexRay.

The only product found in the market, joining CAN-bus and wireless transmission, is *Kvaser*'s *BlackBird SemiPro*. This device is aimed at testing, and evaluation of networks in situations where mobility is

a concern.

In this project, the system developed makes a combination not found on the market of two technologies, the CAN-bus and a *Zigbee* wireless link that will be discussed further ahead in section 2.4.

1.3 System definition

The proposed system comprises two main blocks: a Mobile station and a Base station. The first, is placed on the vehicle and is connected to its sensors through a CAN-bus, a vehicle bus protocol used widely in the automotive and industrial environments. This mobile station has the function of recording locally the data generated in the sensor modules (available through the CAN-bus) and also of transferring this data wirelessly to the base station located off-track. The second has the function of picking up the data sent through the wireless link and presenting it in an attractive and comprehensible way to the user. Besides the "online" operation, the base station is also capable of presenting data from previous logging sessions for analysis. A block diagram of the complete system is shown in figure 1.3.

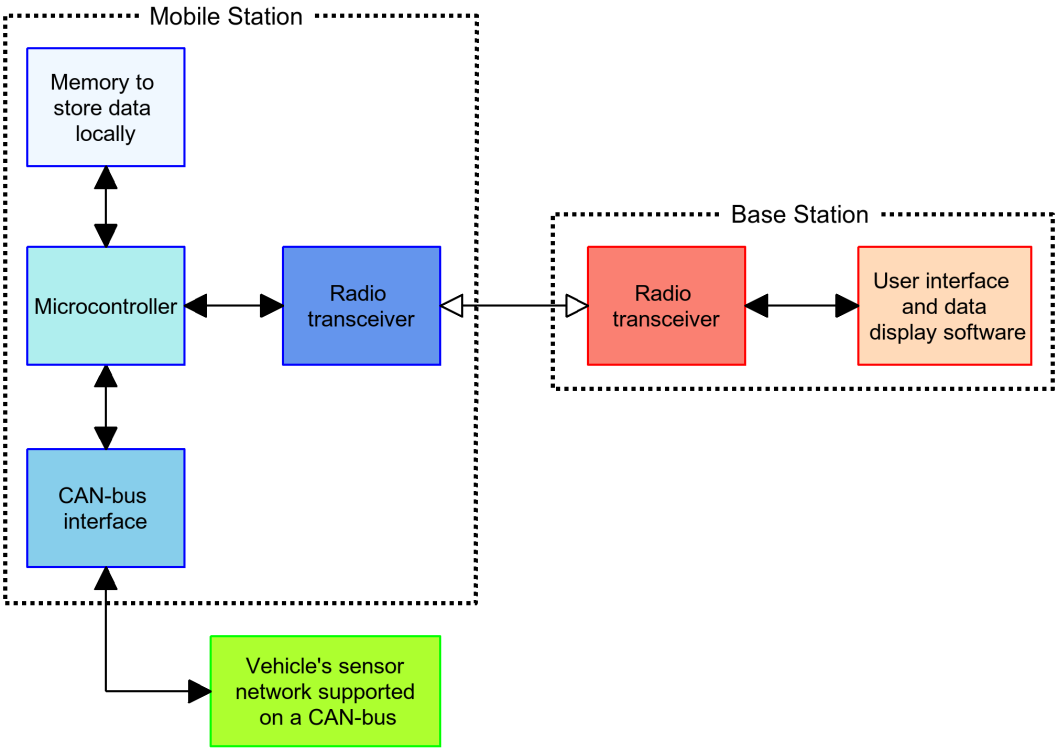


Figure 1.3: The main building blocks of the system and their connection to the vehicle's sensor network.

1.4 Structure of the thesis

This report is divided in four chapters.

In chapter 2 the Mobile station segment of the system is described along with its components, construction and functioning details. It is explained how data available on a CAN-bus is recorded and sent through a wireless link.

In chapter 3 the Base station segment of the system is presented and it is explained how live data is shown to the user. Further options, in terms of data visualization, in the user interface are also explained.

In chapter 4 the performance of the built system is evaluated in two different environments and conclusions based on system usage are drawn.

In chapter 5 conclusions about the CAN-bus logging and telemetry system are made along with a cost analysis and ideas for future improvements.

Chapter 2

Mobile Station

2.1 Overview

The mobile station is the part of the system that is placed on the vehicle. It has a CAN-bus interface, to communicate with the existing sensor network (figure 2.1c), a memory unit to store the data (figure 2.1d), a wireless interface to enable the communication with the base station (figure 2.1e) and a control and processing module to manage its operation (figure 2.1a). Other pieces that compose the system are a real-time clock (figure 2.1b) and the power supply regulation (figure 2.1f)

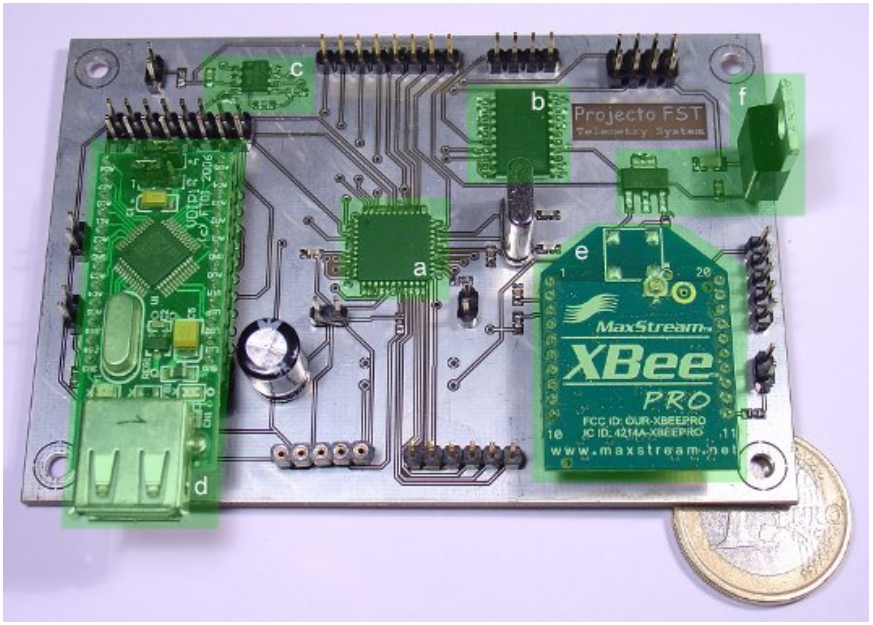


Figure 2.1: The mobile station's board with all its components. (a) Microcontroller, (b) Real-time clock, (c) CAN-bus transceiver, (d) USB host controller module to connect USB drives, (e) *Zigbee* radio transceiver and (f) Power supply regulation.

2.2 Microcontroller and Real-Time Clock

2.2.1 Microcontroller

At the core of the mobile station a microcontroller is used to interconnect and control all the other modules (figure 2.1). The device used is a Microchip's PIC18F4685 8 bit microcontroller. The use of a *PIC* microcontroller in face of other types (AVR, MSP430, etc) is related to the author's previous experience with this kind of devices. The key features for the choice of this device (among others in its family) are the combination of its large program memory of 96 kB and the peripheral communication buses supported - CAN, I²C and USART[1] [11].

The fact that this Integrated Circuit (IC) is available both in PDIP and in SMD packages was of an enormous value to the development of this work. The first because it allows a very quick and effortless design to prototype phase, enabling the development to be done first in breadboard without the need to go right away to PCB layout and the second because it allows a very small footprint on the finished board, which is of great importance in such an embarked system.

Considering the block diagram in figure 2.2, the microcontroller is the device that interconnects all the other modules of the Mobile station subsystem. The Microcontroller Unit (MCU) operates at 5 V and runs with an oscillator frequency of 40 MHz, provided by the external 10 MHz crystal in combination with the internal 4×PLL. Further details concerning the electrical connections of the microcontroller and a schematic diagram can be found in appendix C.1.

As for the remaining components that can be seen in figure 2.2, the CAN-bus and the radio transceivers used will be described in section 2.3 and section 2.4 respectively, the use of a USB host controller will be explained in section 2.5 and the RTC will be introduced in the following paragraph.

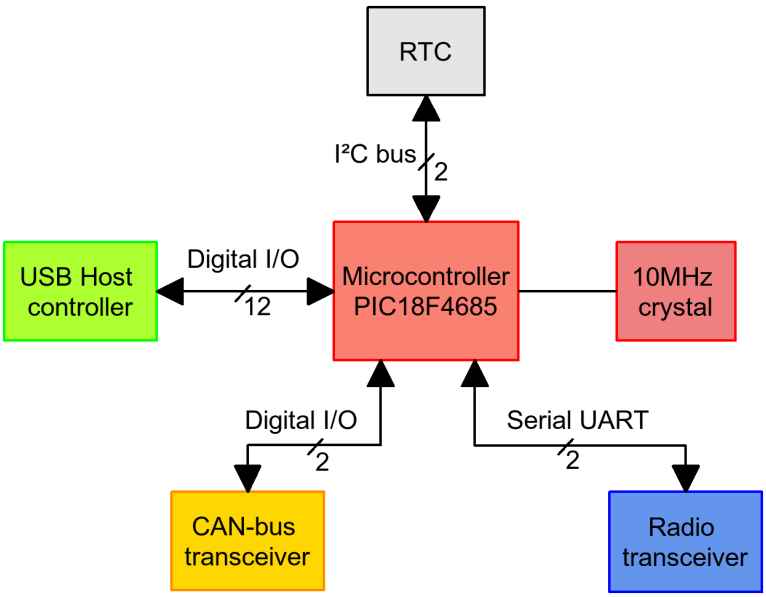


Figure 2.2: Microcontroller PIC18F4685 connections block diagram.

[1] Throughout this report the terms Serial line or port are used to refer to the USART communications channel.

2.2.2 Real-Time Clock

In order to have a time base for the files created on the memory module (section 2.5) a real-time clock (RTC) is added to the mobile subsystem (figure 2.1b). Connected to the microcontroller, using an I²C bus, is a Maxim's DS3232 RTC, which has an accuracy of ±2 ppm and incorporates in its package an oscillator[2], a temperature sensor and age compensating logic, achieving in this way a very robust device that can withstand temperature swings [18].

This device's typical operating voltage should be 3.3 V, but for simplicity of design (level conversions in its bus lines) it is ran at 5 V, which is still well within its maximum ratings[3] [18, page 2].

The DS3232 integrates a battery backup input for continuous timekeeping. A 3 V (200 mAh) coin cell battery (CR2032) is connected to it (figure 2.3). When the supply voltage drops bellow the battery's voltage, the power supply changes to battery mode. In this mode the typical current consumption for timekeeping is 1.5 μA and therefor the cell's life is more than 15 years, which is large enough for this application.

Connection to the microcontroller through I²C bus

As mentioned above, an I²C bus enables the communication between the RTC and the microcontroller (figure 2.3). This is a bi-directional bus that has a Serial Clock Line (SCL) and a Serial Data Line (SDA), with both lines being pulled high by 10 kΩ resistors so that when the bus is free both lines are at the high logic level. All the devices are connected in parallel and each slave device has a unique address which the master uniquely chooses to communicate with. Although it is a multimaster protocol, only one master can be present at any time since the master node is the device that issues the clock on the SCL line.

For this project, the I²C bus master is always the microcontroller, and the bus working speed is set to *Fast Mode*, that is, at a maximum 400 kHz clock rate. Details on the RTC's wiring and connection with the MCU can be found in appendix C.2.

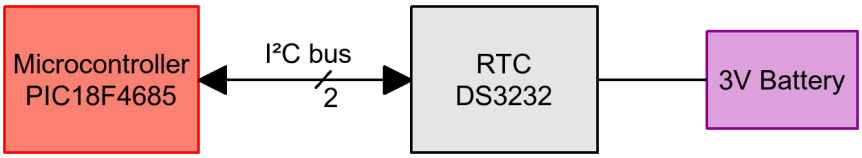


Figure 2.3: Real-Time clock DS3232 connections block diagram.

[2]More precisely it is a temperature-compensated crystal oscillator (TCXO)
[3]Many hours of flawless operation have proven these ratings to be correct

2.3 CAN-bus Interface

2.3.1 Protocol

The Controller Area Network (CAN) is a serial communications protocol developed by Bosch [3] widely used in automotive applications. The CAN communication protocol is a Carrier Sense Multiple Access/Collision Detection (CSMA/CD) protocol. This means that every node on the network must monitor the bus for a period of no activity before trying to send a message on the bus (Carrier Sense). Also, once this period of no activity occurs, every node on the bus has an equal opportunity to transmit a message (Multiple Access). As for the Colision Detection, if two nodes on the network start transmitting at the same time, the nodes will detect the collision and take the appropriate action. In the CAN protocol, a non destructive bitwise arbitration method is utilized, which means that messages will remain intact after arbitration, even if collisions are detected. All this arbitration takes place without corruption or delay of the higher priority message (the node transmitting the highest priority message will continue and the others will wait) [9]. More details on the CAN protocol arbitration can be found in appendix B.1.

Another important characteristic of this protocol is that it is a broadcast or message-based type of protocol, which means that all the nodes receive all the messages (in opposition to an address-based type of protocol), and for specific node addressing to be accomplished, local message filtering must be used [12]. This means that uppon receiving a message, each node decides if the message is kept for further processing or immediatly discarded. The protocol guarantees that messages are accepted by either all nodes or non of them, implementing, for this, several mechanisms to ensure data consistency, error handling and error confinement as is further described in appendix B.2.

2.3.2 Transceiver and configuration

The CAN protocol specification by Bosch does not define the physical layer to be used, so to implement this bus the standard used is the ISO-11898-2[4]. This standard is used for high-speed applications, and it specifies a 5 V differential electrical bus as the physical interface.

In order to interface the CAN-bus controller inside the MCU to the physical medium (figure 2.4), a transceiver is used - Microchip's MCP2551 CAN transceiver (the schematic diagram of this connection can be found in appendix C.3). This IC translates the TTL level signals output by the microcontroller into signals conforming to ISO-11898 up to a speed of 1 Mbit/s [10].

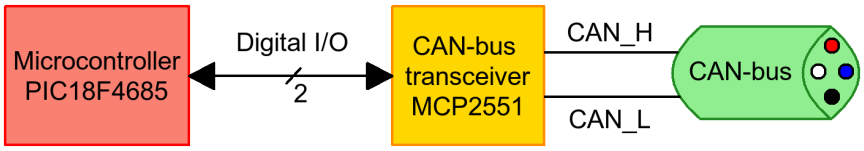


Figure 2.4: CAN-bus transceiver MCP2551 connections block diagram.

For the successful CAN-bus operation all the controllers attached to it must be configured to operate at the same speed and so the symbol timing must be the same, although the clock at which each controller is working can be different. In the case of the CAN-bus (interconnecting all the sensor modules)

[4]Defined by the International Standards Organization (ISO) and SAE.

this system connects to, the speed is set to 1 Mbit/s, the highest speed the protocol can support. By setting the symbol timing registers according to the values in table 2.1 the CAN-bus controller inside the MCU can listen to all the messages on the bus. The way to compute these values and their meaning in terms of bit segments is presented in appendix B.4. The CAN protocol version used in this work is 2.0B, but only standard 11 bit identifier are used in the messages. The CAN-bus message format can be found in appendix B.3.

Table 2.1: Bit timming registers setting for 1 Mbit/s CAN-bus operation.

Register name	**Value**
Synchronization Segment	1
Propagation Time Segment	2
Phase Buffer Segment 1	4
Phase Buffer Segment 2	3
Synchronization Jump Width	0
Baud Rate Prescaler	1

According to physical layer specification ISO-11898-2 the CAN-bus should be fitted with a terminating resistor of 120 Ω on both ends with the purpose of eliminating signal reflections and ensure the bus has the correct DC values [12], as can be seen in figure 2.5.

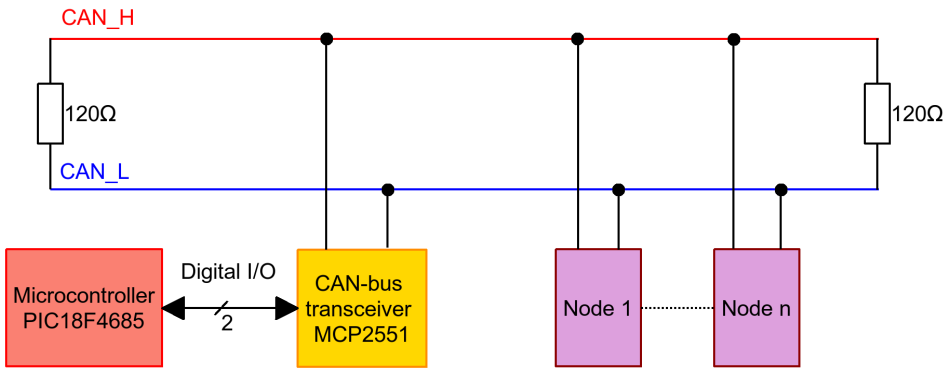


Figure 2.5: CAN-bus physical layer block diagram.

2.4 Wireless Link

2.4.1 Hardware choice

To implement the wireless communication between the mobile and the base station, different technologies were compared. A common starting point to all the solutions taken into account is the usage of a Commercial off-the-shelf (COTS) product, an approach justified by the project scope, time constrain and the quality of products found nowadays. In this comparison the requisites taken into account are a minimum range of 500 meters and a transmission rate over 80 kbit/s[5] and the lowest price possible. The minimum range was set by the distance (plus a margin) the radio must be able to cover both in competion environment and in the practice track - Kartódromo Internacional de Palmela (KIP) (Fig. 2.6), with the last one being the larger, and therefore the constrain.

Figure 2.6: Google Earth image of KIP, the practice track used.

The choice is between the following 6 wireless technologies: *Wi-Fi* (IEEE 802.11) [4], *Bluetooth* (IEEE 802.15.1) [6], *WiMAX* (IEEE 802.16) [5] [20], GSM, UHF radio and *ZigBee* (IEEE 802.15.4) [7]. The usage of *Bluetooth* and *Wi-Fi* modules is set aside because of the shortage of products for a range higher than 200 meters, although the transmission rates are high (11 or 54 Mbit/s with *Wi-Fi* and 1 or 3 Mbit/s with *Bluetooth*). Both *WiMAX* and GSM are eliminated based on the cost of the solutions found, the first due to the very high price of modules (although the transmission rate is high - 75 Mbit/s) and the second due to the pricing over the communications themselves since it uses frequency chan-

[5]The transmission rate minimum value is based on an estimate amount of data produced by 30 sensors, with a resolution of 16 bit, a sampling rate of 100 Hz and considering that this amount is only 60% of the total.

nels allocated to mobile communications. The UHF radio devices have much lower transmission rates (115.2 kb/s) although prices and ranges available are very competitive. The *ZigBee* radios found on the market present higher transmission rates (250 kbit/s) but have a lower range when compared to the UHF radios. The *ZigBee* modules taken in consideration have the best transmission rate/price ratio and a range than confortably meets specifications. In table 2.2 a wireless transceivers comparison chart can be found (or appendix D.2 for an extended version).

Table 2.2: Transceivers' details chart.

Name	**Frequency (MHz)**	**TX/RX (kb/s)**	**Power (mW)**	**Price (€)**
SparkFun LN96	915	9.6	500	68.76
Rfsolutions X8200	900	115.2	500	563.58
Rfsolutions 232C-868F	868	38.4	5	143.24
Rfsolutions TinyOne Pro	868	38.4	500	113.5
Maxstream XTend OEM	900	115.2	1000	138.76
Maxstream XTend Modem	900	115.2	1000	231.78
Maxstream XBee-PRO OEM	2400	115.2	60	27.91
Maxstream XBee-PRO Modem	2400	250	60	84
Aerocomm AC4424-200 OEM	2400	576	200	98
Aerocomm AC4790-1000 OEM	900	76.8	1000	77.5

2.4.2 *Xbee-PRO Zigbee* Radios

The chosen modules are *MaxStream*'s *XBee-Pro OEM*, that can be seen in figure 2.7. Originally meant for use in wireless sensor networks due to their low power consumption and low price, these modules, operating in the ISM 2.4 GHz frequency band, provide a good solution for reliable communication between the mobile and the base station. The key features of the devices, as given by the manufacturer [19, page 5], are presented in table 2.3[6].

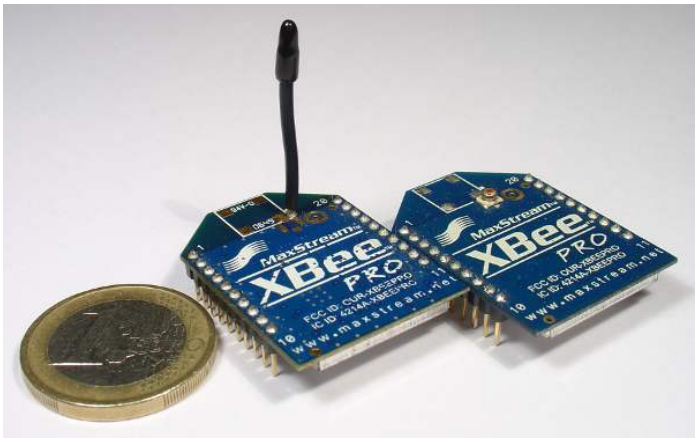


Figure 2.7: *XBee-PRO* transceivers fitted with a *whip* antenna and a male U.FL antenna connector.

[6]The *RF Data Rate* value is the total transmission rate off the wireless link (therefore includes payload and packet overhead) and the *Serial Interface Data Rate* is the transmission rate of the module's interface.

Table 2.3: *XBee-PRO* transceivers specifications.

Specifications	**XBee-PRO**
Indoor/Urban Range	Up to 100 m
Outdoor line-of-sight Range	Up to 1500 m
Transmit Power Output (software selectable)	60 mW (18 dBm) conducted
RF Data Rate	250.000 bps
Serial Interface Data Rate (software selectable)	1200-115200 bps
Receiver Sensitivity	-100 dBm (1% packet error rate)
Supply Voltage	2.8-3.4 V
Transmit Current (typical)	215 mA (@ 3.3 V and 18 dBm output power)
Idle / Receive Current (typical)	55 mA (@ 3.3 V)
Power-down Current	< 10 μA
Operating Frequency	ISM 2.4 GHz band

In order to validate the specifications given by the manufacturer, several figures of merit were obtained with a spectrum analysis. The power/distance ratio was also studied. Measurements were done for the center frequency, the channel power, the bandwidth occupied by a channel and the Adjacent Channel Power Ratio (ACPR), for different power output settings to look for possible changes in performance. Since the signal to measure is only present in bursts (sets of data) the spectrum analyzer is configured to hold the highest sample in the last 200 acquisitions.

The setup used for these measurements is shown in figure 2.8, where the manufacturer's software *X-CTU* (appendix D.1) is used in the personal computer (PC) and the *XBee* development boards are used for module interfacing and power. The software running on the PC makes the *Base Module* send data sets at a certain power level. The data is retransmitted back by the *Remote Module* thanks to the use of a loopback adapter. The need of the second transmitter (remote) comes from software limitations, which makes the first module only transmit a new message when the last one arrived.

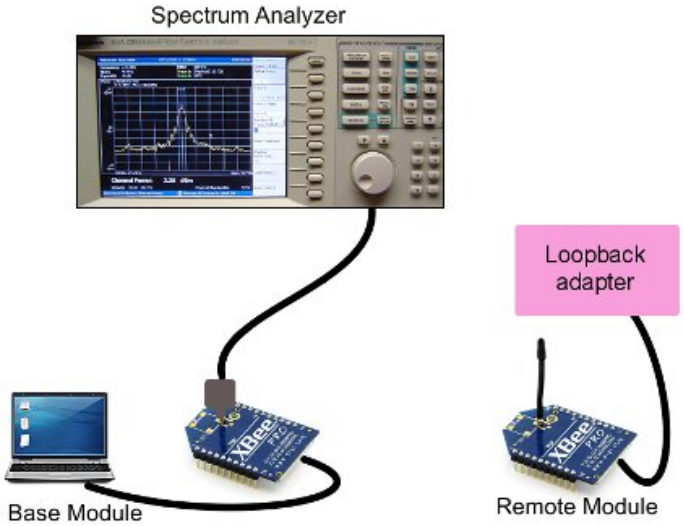


Figure 2.8: Setup for spectral analysis of RF transceivers.

In figure 2.9, a detail of the connection between the *Base module* and the spectrum analyzer, is shown. A U.FL (module antenna connector type) to SMA converter followed by a SMA to type N converter (spectrum analyzer input type) are used.

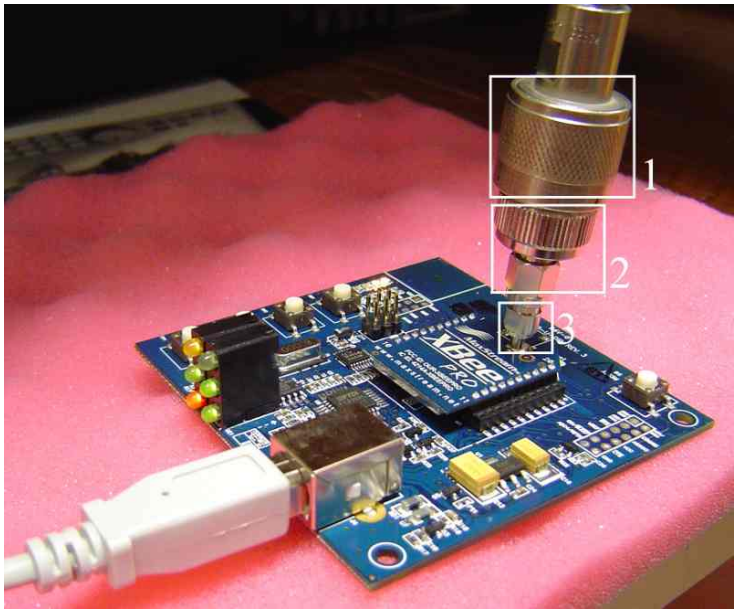


Figure 2.9: *XBee-PRO* radio sitting on the development board. Detail on the connection to the spectrum analyzer: (1) N connector; (2) N/SMA converter; (3) SMA/U.FL converter.

Center Frequency

It's important to measure the center frequency because wrong values can create interference in other channels as well as make it harder for the receiver to pick up the signal. According to [7, page 29], the center frequencies for each channel are defined as

$$F_c = 2405 + 5\,(k - 11)\,, \tag{2.1}$$

in MHz, and for $k = 11, 12, ..., 26$.

The center frequency, of channel 1 (k=12), measured with the maximum nominal power output of 18 dBm (60 mW) can be seen in figure 2.10. The relative error to the value given by equation 2.1 is 0.017%.

Figure 2.10: Center frequency measurement for 18 dBm (60 mW) power output setting.

Channel Power

The channel power level for the *ZigBee* standard is measured with a channel bandwidth of 2 MHz [7, page 266]. If it's too high, it can cause interference and make power consumption rise up and if it's too low, communication can be compromised. In figure 2.11, the measurement is shown for the maximum power output mode (18 dBm). The channel power level is below specifications, with a relative error of 8.89%. The fact that the modules have more than one antenna terminal (so that they can use different types) and that the coupling between the transceiver and the cables may not be perfect, justifies the losses.

Figure 2.11: Channel power measurement for 18 dBm (60 mW) power output setting.

Occupied Bandwidth

The measurement is taken for 99% of the power of the signal. As mentioned above, the channel bandwidth should be 2 MHz. In figure 2.12 the result for the maximum power output level is shown. The result shows a 21% relative error to the expected 2 MHz bandwidth. To measure the occupied bandwidth, the spectrum analyzer looks for the band occupied by a certain percentage of the total power measured. The error obtained is most probably due to a wrong setting of this power percentage value.

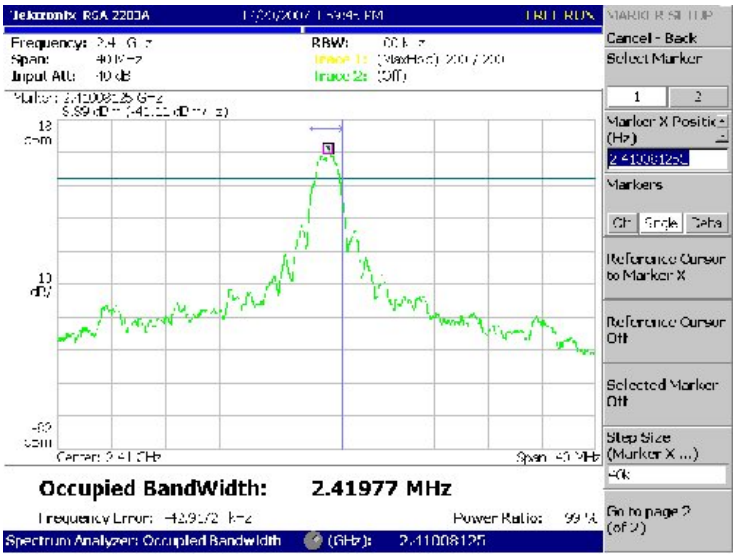


Figure 2.12: Occupied bandwidth measurement for 18 dBm (60 mW) power output setting.

Adjacent Channel Power Ratio

This is a relevant measurement in the characterization of the distortion of subsystems and to determine the probability of the modules causing interference in other systems using different channels. It expresses the relation between the mean power of the adjacent channels and the mean power of the emitting channel. Each channel has a bandwidth of 2 MHz and the channel spacing is 5 MHz, as can be seen from equation 2.1. Looking at the results for the maximum output level (figure 2.13), it can be seen that the ACPR, for the first adjacent channel, is quite low (-41,73 dB) and certainly not enough to affect the integrity of the communication.

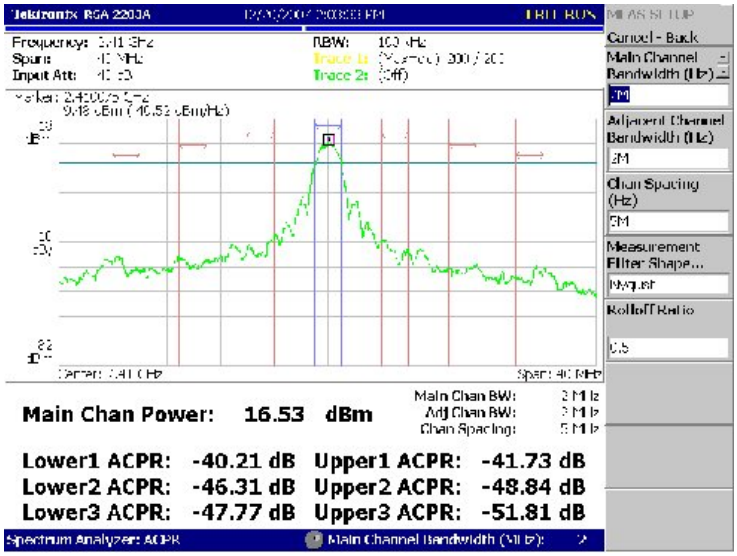


Figure 2.13: ACPR measurement for 18 dBm (60 mW) power output setting.

Power/Distance Ratio

In order to evaluate the quality of the RF connection established by the *ZigBee* radios, a measurement is made of the power, that after being transmitted, ends up being received on the other end of the communication channel - Received Signal Strength (RSS). The higher the received power, the higher the Signal to Noise Ratio (SNR) is and bigger is the probability of success in packet transmission. This leads to the number of retransmissions being reduced which alows a higher transmission rate.

Taking advantage of the radio modules automatically outputting the RSS value, the measurements for power/distance ratio are made, but keeping in mind the indicative character of this output. The tests are run in two different locations, but always for the maximum power output level of 18 dBm. A first test is run at the IST Alameda university campus, using two different antennas, for a distance up to 100 m. The following figure shows the test scheme, where the *X-CTU* software mentioned above is used again. The second test is run in a beach, which allows getting results up to a distance of 300 m. Here the test scheme is different (figure 2.14), and a program in *Matlab* is used to get the RSS output from the transceivers.

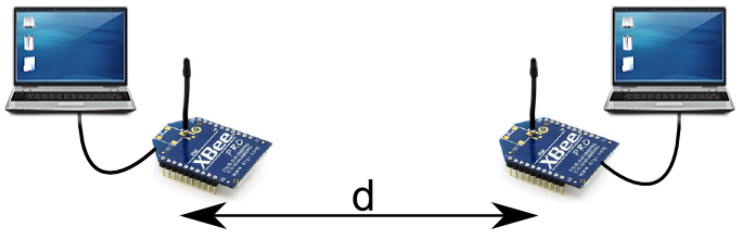


Figure 2.14: Setup for the longer distance test at a beach.

In order to appreciate the results, a comparison is made with the theoretical loss of signal strength of an electromagnetic wave in a line-of-sight propagation through free space - Free Space Path Loss

(FSPL). In this way, having the devices at a distance in meters, d, communicating using a carrier of frequency, f, and with c being the speed of light in vacuum:

$$FSPL = \left(\frac{4\pi d}{\lambda}\right)^2 = \left(\frac{4\pi df}{c}\right)^2 \tag{2.2}$$

which expressed in dB is:

$$FSPL_{dB} = 20\log_{10}(d) + 20log_{10}(f) - 147.55. \tag{2.3}$$

These losses arise from the fact that on the one hand the antenna radiates energy in every direction, decaying in this way with the squared distance, and on the other hand the aperture of the receiving antenna, that varies with frequency and measures how good it is picking up energy from an electromagnetic wave. In these tests the losses on the RF circuits and antenna connections are considered small enough to be neglected and the antennas are thought of as omnidirectional isotropic radiators.

From figure 2.15 which shows the results obtained at IST, it can be seen that a great improvement in the RSS value is obtained with the higher gain antenna (in red) than with the *whip* (in green).

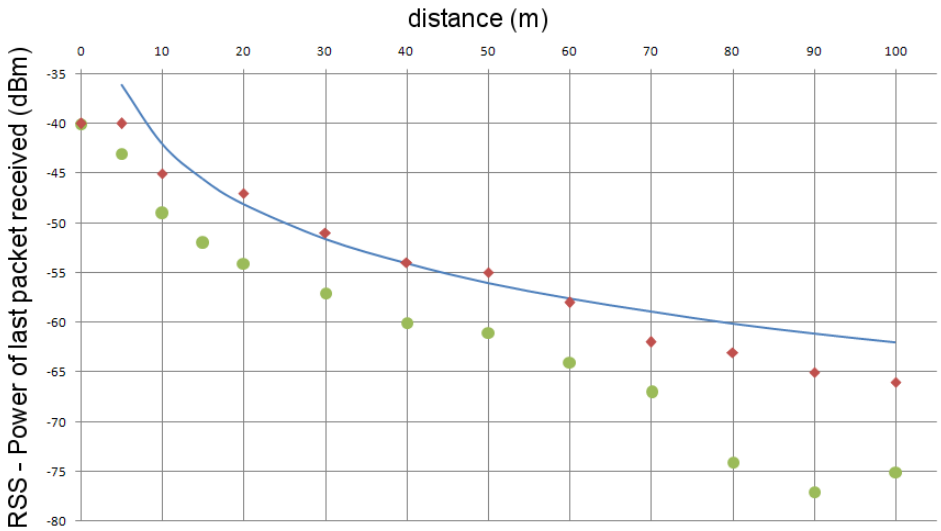


Figure 2.15: Results of the measurements taken at the IST Alameda university campus. $FSPL_{dB}$ in blue. Using the *whip* antenna in green. Using the higher gain antenna in red.

For the beach test (figure 2.16), although the results are quite far from the theoretical value (in blue), still at the tested distance the transceivers are working above the sensibility limit of -100 dBm (table 2.3). The reasons for the values being much lower than the theoretical expectations are believed to be related with the way the test was conducted. The number of samples of the RSS value per distance is very low and since between successive packets there are variations in path and reflections in surrounding objects then, likewise, the power varies a lot. The position of the receiving station, very close to the ground, taking in consideration the irregularity of the beach surface, may have also been a key factor for the difference in the results.

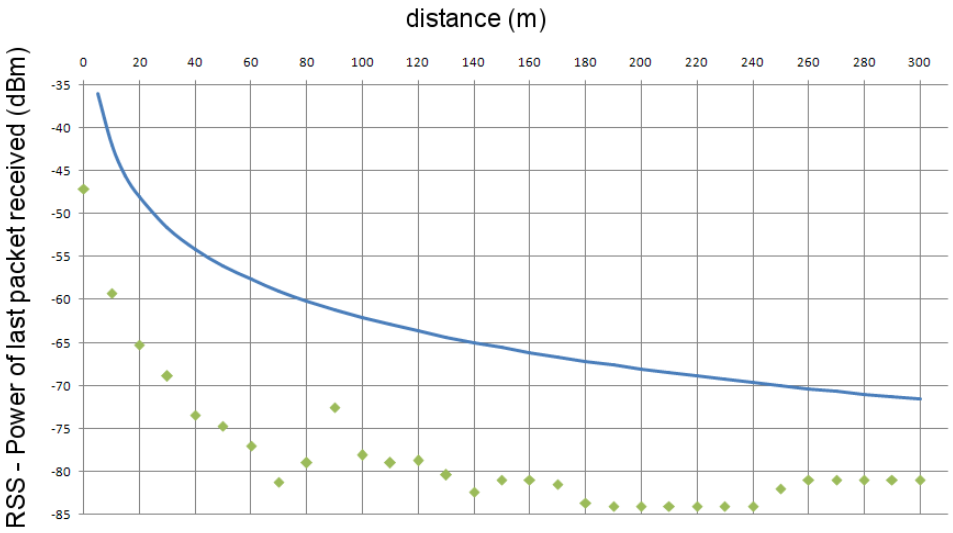


Figure 2.16: Results of the measurements taken at a beach. $FSPL_{dB}$ in blue. Using the *whip* antenna in green.

2.4.3 Connection to the microcontroller and configuration

The RF modules interface with the MCU is done through a TTL level asynchronous serial port (figure 2.17). The baud rate chosen is 115200 bps, the maximum possible for the transceivers according to [19, page 5]. In order to set the baud rate, the vendor's configuration utility is used (appendix D.1).

Since the radios are powered with a different voltage level (3.3 V) than the microcontroller (5 V), there is the need for doing a level conversion in the 2 data lines used for the serial communication. Being these digital connections, a single pull-up resistor is used for the output line[7], and for the input pin, a voltage divider is used to generate the correct voltage for the module pin (schematic diagram in appendix C.4).

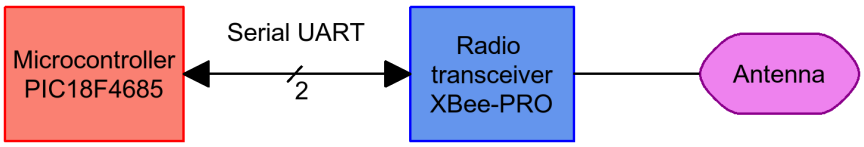


Figure 2.17: *Zigbee* transceiver *XBee-PRO* connections block diagram.

Using the already mentioned *X-CTU* software provided by the manufacturer, the radio transceiver used for the mobile station is configured[8]. The power output setting used is the highest available -18 dBm (60 mW), to make sure the range target is reached and the frequency channel is arbitrarily chosen as 2.410 GHz (the first available channel according to the *Zigbee* specification). The packet addressing mode between modules is broadcasting, which means any module listening in the communications channel will get the data, the module doesn't enter sleep mode and also there are no retries, acknowledge packets nor encription of the messages.

[7] With reference to the Xbee module.
[8] The tranceiver used on the base station is configured in the same way.

As to the antenna, the radio module fitted with the U.FL connector (figure 2.7) is used together with an adapting cable to SMA panel mount lead, where then, a SMA antenna can be screwed, as can be seen further ahead in figure 2.25.

2.5 Data storage

2.5.1 Hardware choice

Since a wireless link is being used, there is the possibility of transmission interruption or fail (range shortage or EMI[9]). To ensure that no data gathered by the sensors is lost, it is necessary to have a storage component.

The choice of a memory element, for the type of system at hand, has to consider several aspects: capacity, speed, price, resistance to the environment, integration effort and also how easy/quick it is to access recorded data. The unit should be able to record the data generated any number of sensors in the CAN-bus network of sensors, as a minimum, the average time of an endurance event - 30 minutes (appendix A.1), and it should have low enough access times so that the maximum recording frequency isn't constrained by it. As for the price and environment resistance it should follow the same line as all the components addressed until now - low price and high resistance to vibration.

In terms of the integration with the other pieces of the system, the storage action should add the least lag to the controller as possible, therefore solutions that minimize the amount of access instructions and avoid complex interface protocols are more suitable. Last but not least, the access to the recorded data should be as friendly and fast as possible, that is, avoiding a lot of operations in order to recover it from the recording medium.

Taking in consideration everything stated in the last paragraphs, the first decision is to use a *flash* type of memory, as it is non-volatile (data is kept without power), it has an acceptable operating speed for this application and great mechanical resistance. Other important aspect is using a FAT[10] file system in the memory device, as it is the easiest way one can access the data in any PC and operating system without any further processing, making it easier and faster for the user.

Considering the options available on the market for *flash* memory technology - SD, CF flash memory cards or USB flash drives, the solution was found to be the combination of a USB flash drive with a USB Host Controller device. The PIC18F4685 has no USB Host Controller[11] capabilities, making it impossible by itself to communicate with a USB Slave device like a memory drive.

2.5.2 *Vinculum* USB Host Controller

The workaround to be able to use a USB mass storage device is to use FTDI's *Vinculum* VNC1L-1A USB Host Controller. This IC is able to, besides handling the USB Host Interface, deal, in a transparent way, with the FAT file structure by including an 8 bit MCU and flash memory [16]. The device communicates with the microcontroller using one of 3 available interfaces: UART, SPI or parallel FIFO[12], and its standard firmware is provided by the vendor. Figure 2.18 depicts the relation between the MCU and the USB drive through the use of the added USB Host controller. This USB drive should be formatted

[9] Electromagnetic Interference. Unwanted disturbance that can affect an electrical circuit due to electromagnetic radiation

[10] File Allocation Table. An architecture to organize files inside memory supported by most operating systems.

[11] The USB bus works in a tiered star topology connecting entities of one of three kinds: *Host*, *hubs* or *device functions*. The system may have several *hubs* connected in series, but only one *Host* is allowed (where the "root" *hub* connects).

[12] First In, First Out. Queue style of buffer where data is handled by the order it appears.

in FAT12, FAT16 or FAT32 file systems with a sector size of 512 bytes.

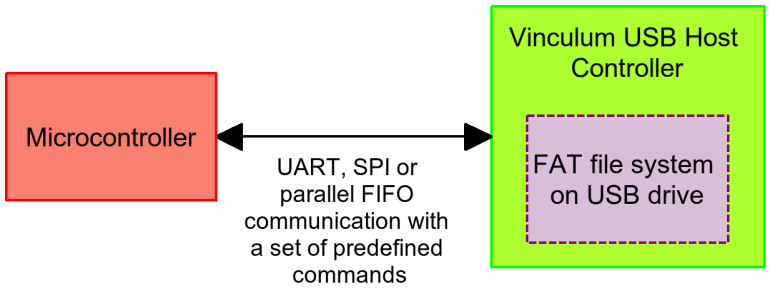


Figure 2.18: Microcontroller and USB drive seamless integration through *Vinculum*.

The *Vinculum* is integrated in the system with the help of the VDIP1 development module. This module, shown in figure 2.19, allows a fast integration as well as a quicker/simpler PCB design stage (section 2.7.1). This module includes a 12 Mhz oscillator crystal, an on-board 3.3 V regulator (*Vinculum*'s supply voltage) and a USB type 'A' socket for connecting the USB drive (USB slave device) [15].

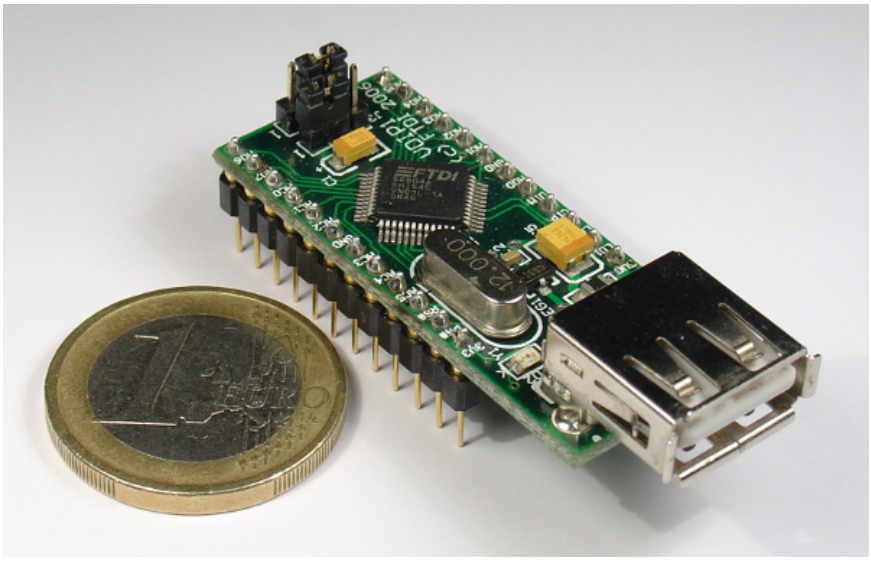


Figure 2.19: The VDIP1 development module that includes the *Vinculum* IC.

Although there is an extra cost introduced by the use of one more module to interface the memory, substantial improvements in operating speed and implementation time are made, as the overhead of handling the FAT structure with the microcontroller (in the case of a memory card) is eliminated, as it is transferred to the dedicated USB Host Controller. Other advantages of this solution are, besides the very high endurance of USB flash drives, their low price and high availability compared to memory cards.

The connection between the microcontroller and the VDIP1 module is depicted in figure 2.20 (and the electrical schematic in appendix C.5). The parallel FIFO interface is used, although it takes a greater number of I/O pins of the MCU (8 data lines plus 4 control lines as can be seen in table C.1), it is the

fastest interface possible since the others use serial communication.

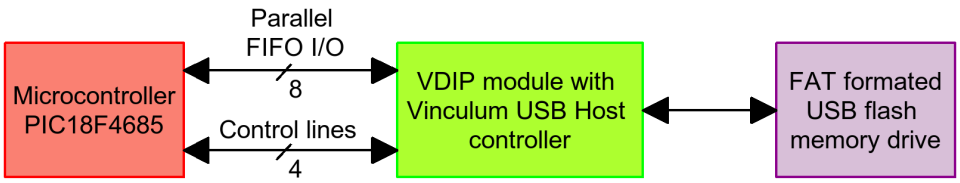


Figure 2.20: VDIP1 module schematic detail.

The *Vinculum*'s FIFO interface operation is configured by the correct setting of the VDIP board jumpers J3 and J4 as can be seen in table 2.4.

Table 2.4: Jumper setting for port selection.

ACBUS6 (VNC1L pin 47)	**ACBUS5 (VNC1L pin 46)**	**I/O Mode**
Pull-Up	Pull-Up	Serial UART
Pull-Up	Pull-Down	SPI
Pull-Down	Pull-Up	Parallel FIFO
Pull-Down	Pull-Down	Serial UART

There are two command entry modes supported by *Vinculum* - *extended* or *short*. These determine the way a command is entered and the format of the replies from the module. In *extended mode*, printable characters are used and commands are typically longer. In *short command mode*, the commands are optimised for programmatical control and have binary values representing the command. For a faster operation, the device's firmware is modified using the vendor's supplied software so that the *short command mode* is set as the default mode of interaction, in a way to reduce the amount of insignificant data exchanged[13].

[13]The details on the *Vinculum*'s interface initialization and interaction are covered in section 2.8.4

2.6 Power supply

2.6.1 Voltage level sources

The prototype vehicle has a 12 V battery that powers all the electrical systems on it and is recharged by the alternator attached to the engine. The mobile station power is supplied by this battery and is available through the same cable as the CAN-bus.

Since two different voltage levels are needed on the circuit, 5 V and 3.3 V, two regulators are used. The main regulator, a LM7805, transforms the 12 V[14] to 5 V (up to 1 A). This regulator supplies all the devices that need 5 V and also the second regulator. This is a TC1264-3.3 that transforms the 5 V in 3.3 V in order to power the radio transceiver. The schematic of the connection of the input power with both regulators is deppicted in appendix C.6. It should be noted, the 7 V voltage drop that take place at the 5 V regulator (LM7805). This together with a peak current supply around 300 mA (as will be seen in the following section) leads to a great power dissipation in this device (2 W). In order to mitigate this effect a heatsink is attached to the regulator as can be seen further in figure 2.25.

During the test of the complete mobile station's circuit it was noted that when the system would be powered without a USB drive and then if the drive would be connected, the MCU would reset. This behaviour was given to the sudden current draw by the USB drive to which the regulator could not respond and caused a drop on the MCU supply voltage enough to make it reset. To mitigate this fault a high value capacitor was added to the system as can be seen in the block diagram of figure 2.21.

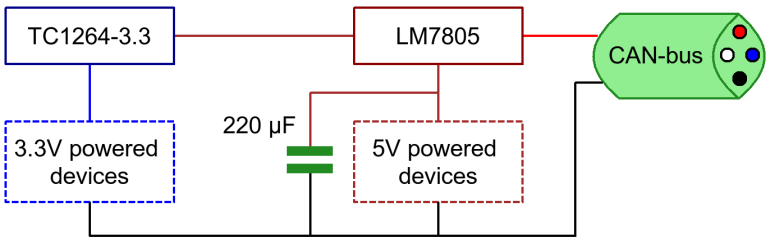


Figure 2.21: Power supply block diagram.

2.6.2 Power consumption

The power consumption by the mobile station was measured in 2 different states. In the *Standby* state, when the system is waiting on orders from the base station, the DC current, as measured with a multimeter, is 145 mA which is slightly higher than expected since the major contributors should be the MCU, that according to [11, page 424] draws typicaly 28 mA (and maximum 44 mA), the *Vinculum* that needs 25 mA [16, page 14] and the *XBee-PRO* radio transceivers, that are configured for no sleep, and so their power consumption, from [19, page 8], is 55 mA[15]. The difference between the expected value and the result for the current consumption in this state is believed to be related with both the USB Host

[14] At full charge the battery's voltage is higher than 12 V for a healthy battery, being able to reach over 13V. This is not a problem for the voltage regulator, that according to [8, page 2] can accept input voltages up to 35 V.

[15] According to [19, page 22] when the modules are configured for no sleep, their power consumption in idle mode is the same as in receive mode.

controller module circuits and the USB memory drive.

For the measurement when the station is in the *Log* state, a different method had to be used in order to be able to characterize the comsumption in the most power intensive phase, that is when a message shows up in the CAN-bus and it is recorded in the memory and sent through the wireless link. In this way, the setup shown in figure 2.22 was used. A 0.1±5% Ω resistor was fitted in series with the positive supply line and the voltage drop between its ends was observed in the difference of 2 channels in an oscilloscope.

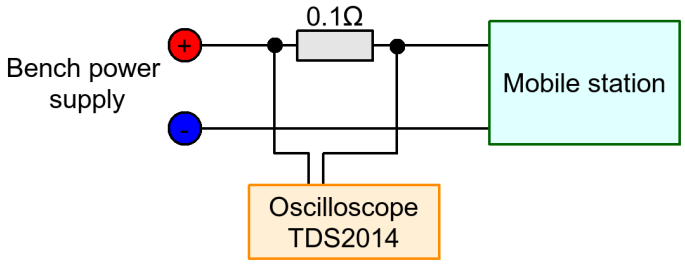


Figure 2.22: Setup used for the mobile station's current consuption measurement.

In figure 2.23 it can be seen there is a voltage drop across the ends of the resistor of 30 mV. From this, the value of the current burst is computed to be 300 mA, a value that is due to the high current consumption of the radio transceiver, according to [19, page 8] of 215 mA, while it is in transmit mode.

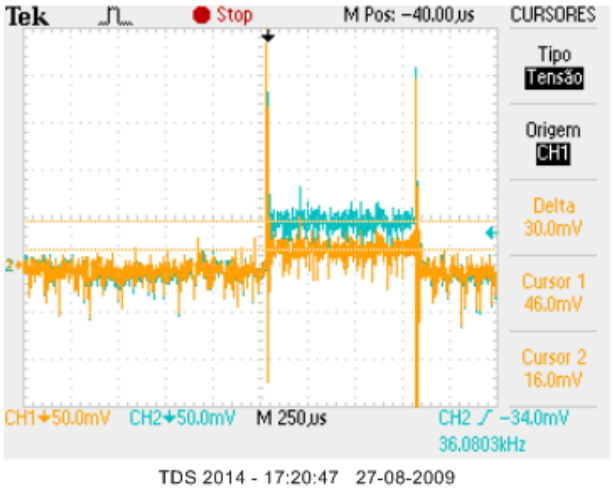


Figure 2.23: Osciloscope image of a current burst that happens during transmission of a through the wireless link.

2.7 Prototype construction

2.7.1 Printed Circuit Board layout

The layout of the mobile station's printed circuit board was done in *Altium Designer*, and it's design had the size, the easy assembly and the high versatility of the prototype as main concerns. The size was reduced by using SMD components as much as possible but still keeping enough space for an easy assembly, as well as for the wires and connections of the board to it's exterior connectors. The versatility and expandability of the board was assured by providing access to some extra wireless radio's pins (RSS signal and 3.3 V regulated supply), to all the microcontroller's ports, and also by fitting extra placement holes for "big" capacitors (seen before in section 2.6.1).

In order to reduce noise all the space between traces is filled with ground plane and extra care is taken in the routing around the RTC, following the recomendations of [18], no signal traces are run underneath the device. The produced board can be found in figure 2.24, and the layout details and drawings can be found in appendix F.

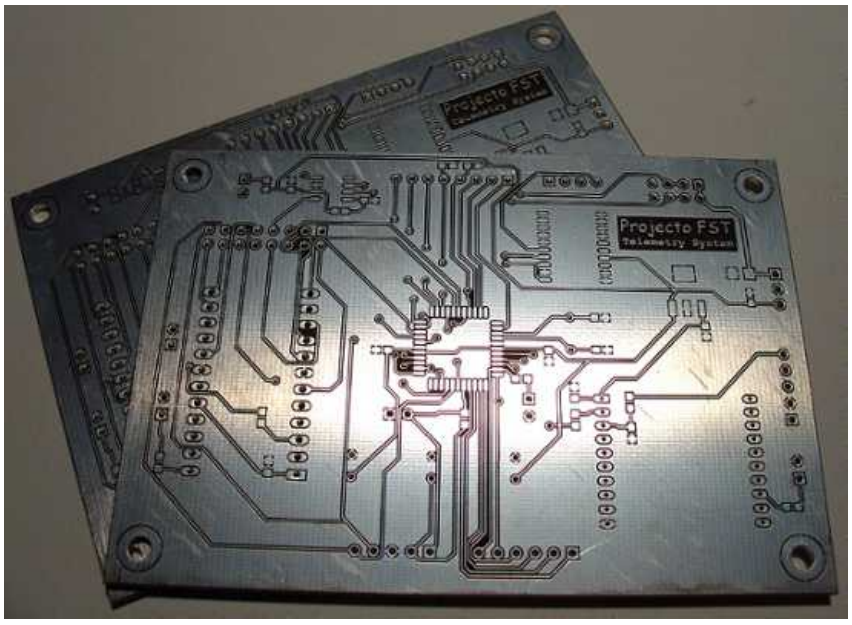


Figure 2.24: Produced mobile station's board prototype.

2.7.2 Enclosure

To house the PCB board, a commercial box was modified in order to support the circuit's connections to the CAN-bus, the USB memory drive, the antenna and the power switch. The CAN-bus cabling running across the vehicle uses IP67[16] sealed connectors, so a flanged plug is mounted on one of the enclosure's side panels for superior resistance and isolation as can be seen in figure 2.25. In the same figure the extra aluminium piece that was fitted to the bottom so that fixation holes could be added can

[16] Ingress Protection rating. The first number concerns solid materials and the second liquids: 6-Totally protected against dust, 7-Protected against the effect of immersion up to 1 m.

also be seen. These provide safe holding points when the unit is placed on the vehicle.

Figure 2.25: Final assembly of the PCB board inside the enclosure fitted with cables to the connector and power switch. The transceiver adapting cable and the main regulator heatsink can also be seen.

Since this system is going to be used by other people, an informative sticker was added. It provides basic information on power and signal connections, as well as, antenna type, for the user (figure 2.26).

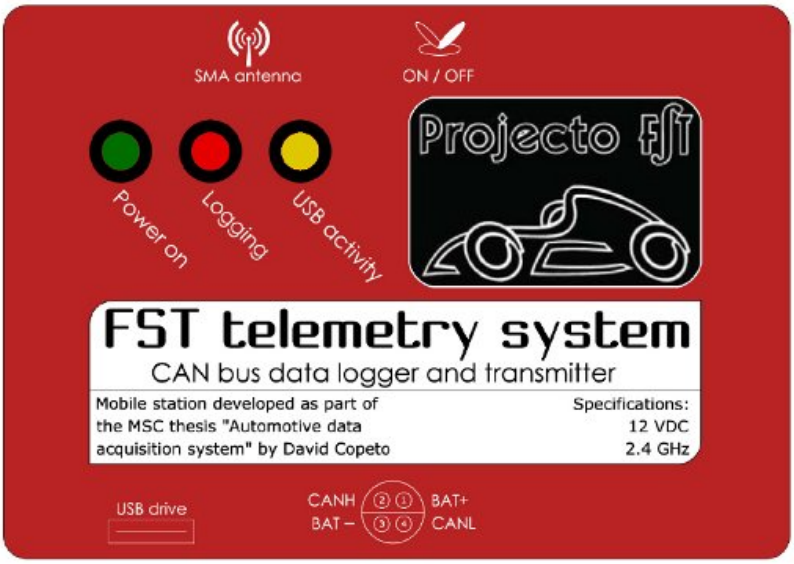


Figure 2.26: Mobile station's enclosure sticker.

To easily find out the state in which the mobile station is, 3 LEDs were fitted to the top of the box. In table 2.5, details on the LEDs working conditions can be found. The *Logging* and *USB activity* LEDs should be taken in consideration specially when turning power off in order not to loose any data.

Table 2.5: User information LEDs' behaviour.

LED	**Information**
Power on	When On, station has power
Logging	When On, station is in *Log* state
USB activity	When blinking, warns for USB memory drive usage

The finished mobile station's prototype can be seen in figure 2.27.

Figure 2.27: Final mobile station's prototype.

2.8 Mobile station Software

2.8.1 Overview

The program state diagram running in the mobile station's microcontroller is shown in figure 2.28. At power up, there is first an initialization stage. The MCU's ports, UART communication channel and the CAN-bus controlling hardware is configured, the *Vinculum* USB Host controller is initialized and the presence of a USB drive is checked for. The program then stays in *Standby* till a *Start logging* command (table 2.7) arrives through the wireless link. From here, a new file is started on the USB drive and the program enters the *Log* loop, being interrupted for each arriving message: messages on the CAN-bus or commands from the base station on the serial line (coming from the radio link). The interrupt routine for the CAN-bus takes care of the incoming messages, putting together a data frame from the meaningful CAN message data and sending it both to the storage device and the base station (through the radio link). If a *Stop logging* command (table 2.7) is received through the serial line, meaning the user issued a stop order in the base station software (covered further in section 3.3.3), the file (on the USB drive) is closed and the environment is reset for a new session. In the following paragraphs a deeper insight into the main routine of the MCU's software is presented, along with the interrupt routines and the MCU's peripherals access routines and implementation details.

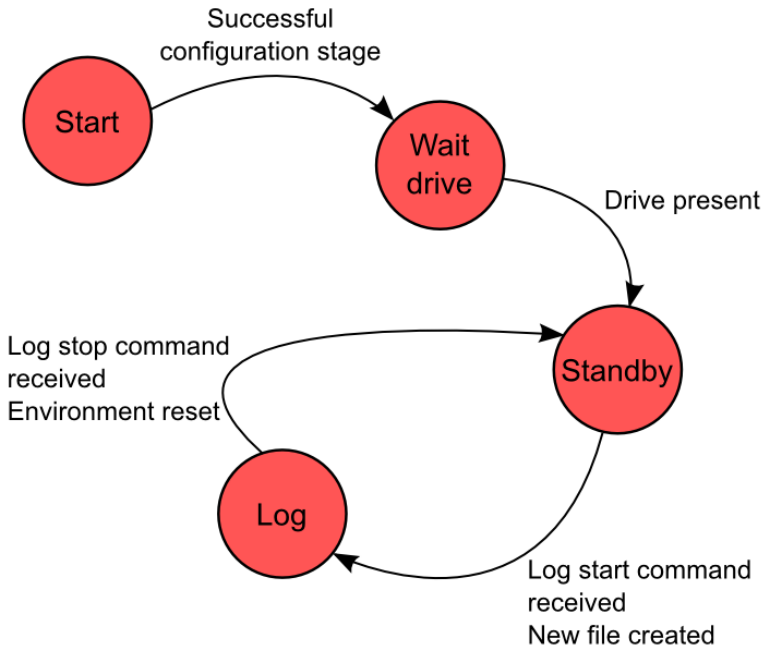


Figure 2.28: State diagram of the mobile station software.

2.8.2 Main function

The main routine of the microcontroller's software, starts by configuring the ports and initializing the peripheral devices connected to it (figure 2.2). The CAN-bus controller registers are configured according to table 2.1 and at this point its interrupt routine (subsection 2.8.6) remains inactive. The USB host controller is also initialized, and the USB drive is detected. If a drive is not present, the MCU holds, querying the *Vinculum* module until a memory drive is found. Next, the mobile station, receives through the serial line the logging session name, chosen by the user at the base station's software startup wizard (section 3.3.3), which is used to build the names of the files created on the USB memory.

After the configuration stage, the serial interrupt is enabled as the station enters the *Standby* state. Here, as can be seen in figure 2.28, the software holds until a *Start logging* command is received from the base station (the available commands from the base station are presented further in section 2.8.5). When this happens, a file name is built using the session name truncated to a maximum of 6 character and a 2 digit number[17] (summing up to a maximum of 8 characters for the name - limitation imposed by the *Vinculum*). With this name, a text file[18] is created on the USB drive (section 2.8.4), then the CAN interrupt is enabled and the *Logging* state warning LED (section 2.7.2) is turned on. With every change in state and successful completed operation in the mobile station, a feedback message (section 2.8.7) is sent to the base station, so that problems can be spotted by the user.

During the *Log* state, the arrival of messages on the CAN-bus triggers the CAN interrupt routine (section 2.8.6) or data arriving through the serial line causes, on its turn, the serial interrupt to be called. When a *Stop logging* commands is received, the *Logging* LED is turned off and all the interrupts are disabled, in order to prevent the user from starting a new logging cycle while the mobile station is not yet ready. The file on the USB drive is closed, all the environment is reset and the serial interrupts are made active again. The system is now ready for a new logging cycle. The flow diagram of the main function can be found in appendix E.2.

2.8.3 Real-Time Clock setup and access routines

The real-time clock value is read whenever a new file is created on the memory drive. There are 4 functions related to the RTC: toBCD(), fromBCD(), setTime() and readTime(). The first two, are auxiliary functions that convert decimal numbers between binary and BCD, and the last ones enable the configuration of time registers and current time reads respectively. In order to communicate with the DS3232 the address used on the bus should be 1101000 as found in [18, page 17].

As mencioned already in section 2.2.2, the system has a battery so that continous timekeeping is possible. In this way, the time on the RTC only needs to be set once (as long as the battery remains connected). For this, function rtc_setTime() is used once, being removed from the program in regular operation[19]. The flow diagrams of the RTC functions can be found in appendix E.1.

[17] The file names generated in consecutive logging cycles of the same logging session include a consecutive numbering.

[18] The text (.TXT) file type is the extension chosen for the files stored in the USB drive for their simplicity and universal acceptance in various operating systems.

[19] Since the time setting is done in code, after adding the function to the main routine, programming and running the microcontroller, it needs to be removed, otherwise it would be executed on every reset of the device.

2.8.4 Data storage routines

The communication between the MCU and the USB Host controller through the parallel FIFO is supported on a whole set of functions that had to be written from scratch. These can be separated in 4 groups, according to the type of interaction, specified in the following table.

Table 2.6: *Vinculum*'s functions by interaction type.

Basic	**Initialization**	**File open**	**File Write**
readFIFO()	vinculumInit()	vinculumOpenFile()	vinculumWriteFile()
writeFIFO()	vinculumDriveStart()	vinculumWriteFilePreamble()	vinculumWriteBlank()
	vinculumDriveStart2()		vinculumWriteFileEnd()
	vinculumWaitDrive()		vinculumCloseFile()

The *Basic* set of functions implements the parallel setting and reading of bits across the 8 data lines (PORTD of the MCU) according to the parallel FIFO protocol timing rules stated in [15, page 7].
For the first interactions functions, group *Initialization*, the functions vinculumInit(), vinculumDriveStart(), vinculumDriveStart2() and vinculumWaitDrive() get the firmware version running on *Vinculum*, read the drive's detection messages and wait for a drive to be inserted, respectively.

Focusing on the interaction set *File open*, 2 functions are used to perform this action. The first, vinculumOpenFile(), is used to open a file for writing by using the *Open File for Write* (OPW) command of the *Vinculum* firmware[14, page 31]. This function passes the name and date/time information of the new file to be created. After this command, for each data set to write, the command *Write To File* (WRF) of the *Vinculum* firmware[14, page 31] should be used to pass the data (and it's size) to write on the file. Now, in order to reduce the time it takes to write data to a file, instead of performing a WRF command each time there is data to be written, a single WRF instruction is sent right after the file is opened. This way, a single WRF is done minimizing the time spent in writes, as there is no need to wait for *Vinculum* to accept each operation. There is, though, a small drawback to this technique, since with every WRF command the size of the data set must be specified. The problem is that at the beginning (when the file is opened) there is no clue on what is the amount of data that will be written to the file. To overcome this, the size of the file is always set to the same figure: BYTES_TO_WRITE, a constant defined on the microcontroller's program[20]. The function vinculumWriteFilePreamble() sends the WRF command to *Vinculum* with the data size to write as an argument.

In the last group of functions, *File write*, to interface the USB controller, the vinculumWriteFile() function is called every time a new data set is ready to be stored in the opened file. These data sets can have any size due to the use of the writing technique mentioned above. There is the need, therefore, to check if there is still enough space for the data set before vinculumWriteFile() is called. That check is done in the function called by the CAN interrupt, as is further explained in section 2.8.6. If the space available to complete the size specified at opening time (BYTES_TO_WRITE) is not enough for the data set "at hands", space needs to be filled up or else the USB controller will not allow the file to be closed. To do this, function vinculumWriteBlank() fills the remaining space with null data.

[20] Further ahead in the *File write* set of functions, will be explained how the problems with setting a constant file size are overcome.

If during the log cycle, the file size is reached, a new file must be created in order to continue the data recording without noticeable interruption. This is accomplished by forcing a new file to be started without the need for the base station control software to intervene. In subsection 2.8.7 the mechanism used to signal this situation to the base station is explained.

When a log session ends, and the file needs to be closed, functions vinculumWriteFileEnd() and vinculumCloseFile(), read the *Vinculum*'s response to the WRF command and send the *Close File* (CLF) command respectively.

The flow diagrams for all the *Vinculum* routines can be found in appendix E.4.

2.8.5 *Xbee-PRO*/Serial interruption execution

This interrupt routine is triggered whenever data arrives (from the base station) on the serial line connected to the radio transceiver. The serial line is configured to work at a rate of 115200 baud with 8 bits, 1 stop bit and no-parity. As can be seen in table 2.7, only 2 different actions can be triggered with this interrupt routine. If the character 'g' is received, the mobile station changes its state to *Log* and a message is put on the CAN-bus for all the nodes in the network to reset their timers, if instead a 's' arrives, the *Log* state is interrupted. In appendix E.14 the flow diagram of the serial interrupt function, isrSerial(), is deppicted.

Table 2.7: Actions triggered by the serial line interrupt routine.

ASCII character received (hex value)	**Command name**	**Action**
g (0x67)	*Start logging*	Change state to *Log*
s (0x73)	*Stop logging*	Change state to *Standby*

2.8.6 CAN-bus interruption execution

The filters on the MCU's CAN controller are set so that messages comming from nodes with any CAN_ID are accepted, as well as any kind of message type. In this way, everytime a new CAN message is put on the bus, isrCanrx1() is called. In order to prevent reentrancy, first the CAN-bus interrupts are disabled while this interrupt routine is being executed. Then the message is fetched from the CAN controller buffers and, if there is still space in the file (BYTES_TO_WRITE hasn't been reached) data is recorded on the USB drive by calling vinculumWriteToFile() (seen above in section 2.8.4) and also sent to the base station through the wireless link by calling sendFrame() (covered further ahead in section 2.8.7). If the file has no more space, a "forced" log stop condition happens, to which follows signaling to the base station and an immediate closing of the file and opening of a new one (in the same way it has been described above in section 2.8.2) so that logging can continue. The flow diagram for the CAN interrupt routine can be found in appendix E.15.

2.8.7 Message format and feedback command set

The messages (or frames) sent from the mobile station to the base station through the wireless link have always the same size and format although they can have different types. The messages can either be data frames or feedback frames. Data frames are sent when the mobile station is in the *Log* state, and they contain the meaningfull fields of the CAN messages received - the CAN_ID of the sensor and the CAN 4 byte data field[21], plus a frame number. The feedback frames use reserved CAN_ID numbers to signal *Start*, *Info* or *Error* conditions to the base station.

The frames are composed by 7 bytes and their format is deppicted in figure 2.29. The first byte, has the ASCII character 'F' indicating the start of frame, the following byte holds the data frame number (only for data frames) and the next byte is the CAN_ID field. The remaining 4 bytes, hold the CAN data when the message is a data frame.

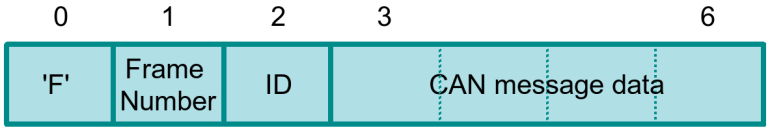


Figure 2.29: Format of the messages sent from the mobile station.

Having limited the CAN_ID field to one byte, translates in only 256 different identification numbers being available (from the 2048 available using the 11 bits of the standard identifier field of the CAN-bus message). From these, ID numbers above 97 are reserved for feedback frames. In table 2.8, the available feedback frames are presented.

[21] Although the CAN protocol allows up to 8 data bytes in each message, in the CAN-bus network of sensors this work connects with, only 4 bytes are used.

Table 2.8: Set of feedback frames used by the mobile station.

ID number	Type	Description
98	Info	Normal Log stop
99	Info	Forced Log stop
100	Start	Start of program
101	Start	Configuration of CAN communication
102	Start	Date/Time set
103	Start	*Vinculum* start-up
104	Start	Start of standby cycle
105	Start	Start of new log file
106	Start	Start of log cycle
107	Start	Drive start
108	Info	Extra data
109	Info	Drive present
110	Info	Drive not present
111	Info	File created/opened
112	Info	Ending file
113	Info	File closed
114	Info	Log start acknowledge
115	Info	Sensor info
116	Error	*Vinculum* can not change to SCS
117	Error	File can not be created/opened
118	Error	File could not be closed
119	Error	Other error
120	Error	CAN error
121	Info	CAN time reset

The feedback frames are used to provide feedback to the base station on which state the mobile station software is in, as well as, give information on the result of the interaction with the peripheral devices and warn on possible error conditions. The function sendFrame() is the routine used for sending frames through the serial line and its flow diagram can be found in appendix E.16.

Chapter 3

Base Station

3.1 Overview

The base station is the part of the system that is located off-track (figure 3.1). Its hardware is formed by a wireless transceiver and a PC. The wireless transceiver in the base station completes the RF link between the two parts of the system as seen already in figure 1.3. The PC runs the user interface software to the system, allowing both “online” data collection and “offline” data analysis. The connection of the radio module to the computer is done with the help of the same *XBee* development board used for device characterization (figure 2.9).

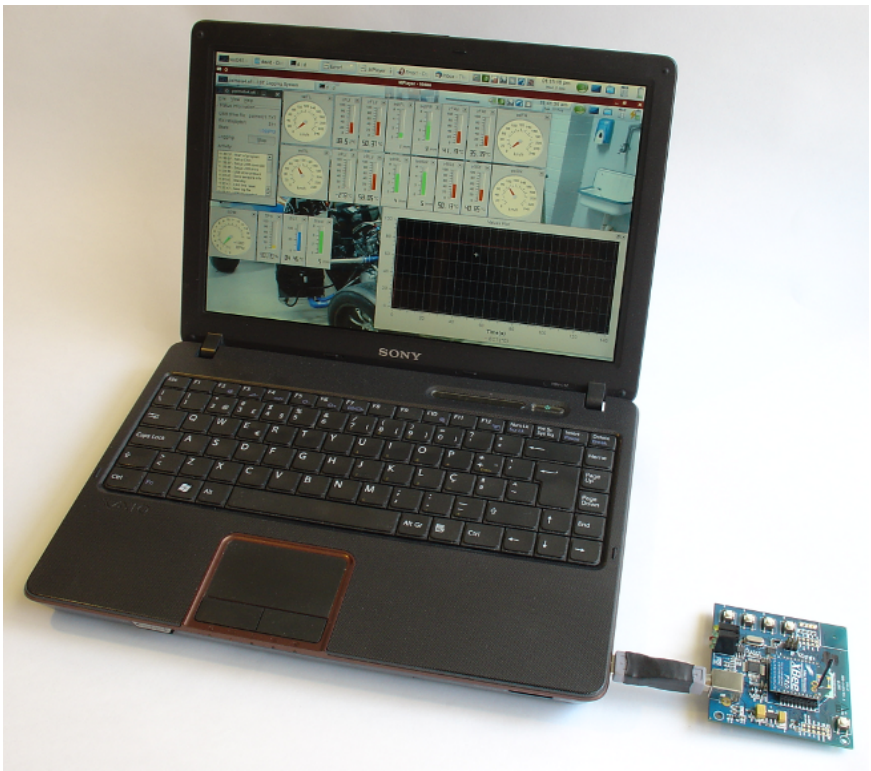


Figure 3.1: Base station.

3.2 Programming language choice

The first choice when developing the system's user interface, was the programming language. This is an important step as the chosen software tools can condition the development speed and freedom as well as the application's final overall performance. In this way, the chosen "software package" must, above all, include libraries to access the serial port (as the interface with the radio module is done in this way), have graphic libraries that allow the creation of a fairly complex graphical interface and also allow the development in an open-source OS[1].

The options taken in consideration were the use of JAVA, Matlab, Labview and C. The first two, although managing almost all the requisites and already including complete graphic libraries were set aside beacause the applications tend to become "heavier" (speedwise). Labview, was thought as not being appropriate to the kind of application and C, although producing the "lighter" applications lacks object-oriented programming. In this way the chosen base language was C++. Being a high level language with low-level functionalities, allows a good performance, and is a good tool to implement the versatility desired for the application.

As to the graphical interface, it is developed using the Qt Framework together with the Qwt libraries. Qt is multiplatform graphical package, which means that software migration between different operating systems is easier, and it includes a large collection of graphical widgets[2] to incorporate in applications. Lastly, the Qwt libraries are used to add a few visualization widgets that make showing data to the user more efficient and enjoyable.

[1] The experience of the author with certain languages is also a very important point, although, in the case of this work, not decisive.

[2] A graphical widget is the general name of any item that composes a graphical interface, like buttons, windows or icons.

3.3 Software

3.3.1 Objective

The general purpose of the software running on the base station is to give the user an interface to the system. With more detail, the graphical interface should be an easy and attractive tool able to plot incoming data, have a versatile and reconfigurable structure so that it allows different sized CAN-bus networks to be monitored, accept different sensors and sample rates, be a tool to perform analysis on previously gathered data and also be able to recover data stored on the mobile station's USB memory device.

3.3.2 Structure

The application is structured according to figure 3.2. Since it is possible to configure the software for the kind of session and network it will work with, an important piece is the startup wizard. This pre-application configuration tool guides the user into the sort of session he wants, asking for all the information needed to build the environment. This results in the main application window and the database being built. Deppending on the type of session the database is filled up with data values, in the case of an *Offline* session, or is empty, in the case of an *Online* session. Likewise, the data source and structure of the main window also depends on the type of session. The data is completly loaded at startup if its source is the hard drive or the USB memory drive, or, in the case of an *Online* session it is received by the serial interface "block" comming from the *Zigbee* radio. After the main window is built, the user has the possibility to change a few details in the visualization of the items and also control the state of the mobile station.

3.3.3 Functionality details

In the following paragraphs the functions and characteristics of the core pieces of the software are presented. In appendix G.1 the class diagram of the base station software is deppicted.

Wizard

The wizard is composed of a series of input fields designed to help start the application (figure 3.3), and cannot be skiped. The user must complete the forms in order to properly build the application for the desired purpose, although in an *Online* session some of the information entered can be later modified.

When the option for a *Create...* (*Online session*) is made, in a first page the name of the session is entered and the option to read a CAN-bus network composition from a previously created session file is presented (figure 3.4). Then in a second page the CAN-bus network arrangement is requested. In order to properly build the main window, the user provides a name, the CAN-bus ID and type of each sensor, and also choses if the sensor data is to be viewed in the plot area, as deppicted in figure 3.5).

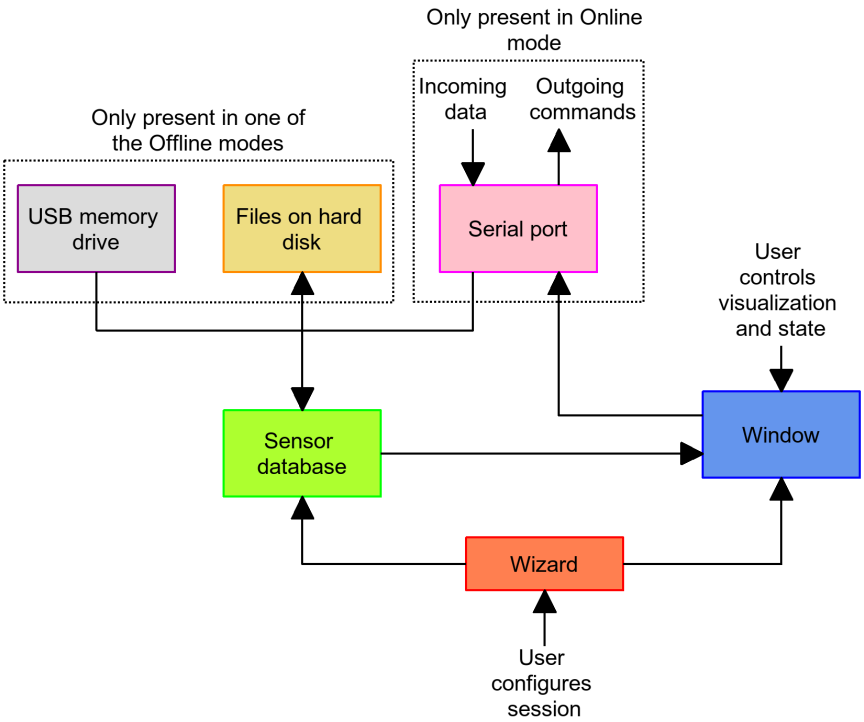


Figure 3.2: General structure of the base station software.

For the case of a *Load...* (*Offline session*), the user simply points to the session file (the file types are refered further in the Database paragraph) and the main window is built without further information input (figure 3.6).

If the third option is selected, the *Retrieve...* (*Offline session*), the user points the USB memory's file to be read and then adds the information regarding the type of each sensor found in the file, which is necessary for a correct loading and plotting of the values (figure 3.7).

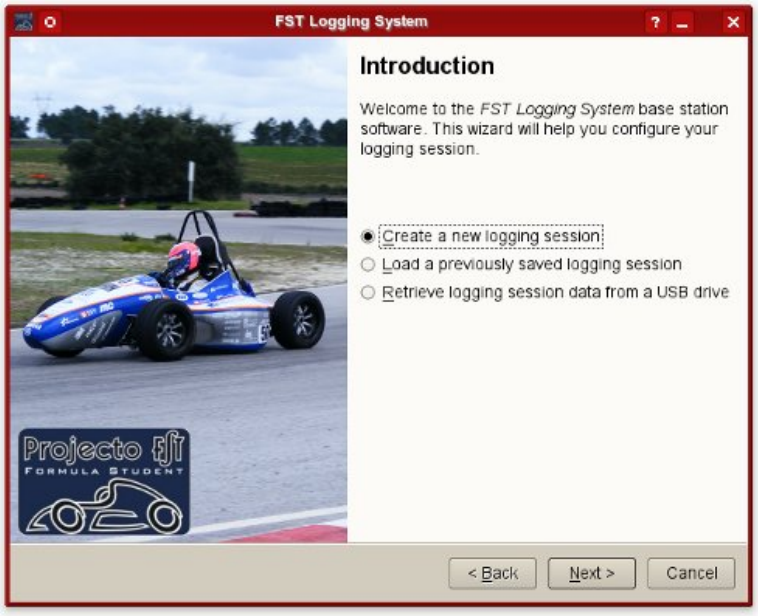


Figure 3.3: Wizard start page.

The application's wizard is accomplished using a series of classes, one for each page in it. Worth of reference is the method *accept()* of class *startWizard* which is called when the user presses the *Finish* button. Here, for every type of session, the database is built and for the *Offline* type of sessions, data values are read from files and the database is built.

Database

The structure holding the data is organized as a list of sensors. For each sensor, besides the data values, details on name, CAN-bus ID and type are kept, and whether the values should be ploted or not (*Online* session). There are 8 types of sensor available: Tire temperature, Throttle position, Steering position, Wheel speed, Suspension displacement, Engine coolant temperature, Engine RPM and Sine Wave (for testing purposes). For each type a different display widget exists as seen further ahead.

This database is always built based on the inputs of the user to the wizard and can be changed while the program is running, in the case of an *Online* session, and is "filled up" with the incoming data from the serial interface. For the *Offline* sessions, it is completly loaded with data values at startup.

In order to save the data on the PC's hard drive, two types of file are used. One, the .sfl file, registers all the parameters concerning a logging session, namely date and time, number of sensors, number of values per sensor and number of runs (explained further in the *Online* Window paragraph), among others. The second type is a .txt file per each sensor logged, that stores all the data values received as well as the sensor details. Example contents of these files can be viewed in appendix G.2.

Two classes are used for organizing data, the *sensor* and the *dataBase* classes. The first has just one method, to create objects that populate the list existing on the second. Notice that for each *sensor* object created, a new output file (to store the data values) is created too.

Figure 3.4: Name and previous session reading in a new *Online* session.

Figure 3.5: Configuration of the CAN-bus network structure in a new *Online* session.

Window

***Online* session** In this session the main window (figure 3.8) is built dinamically, in acordance with the options entered by the user, which means that not all sensor display widgets are viewable everytime, only if they are being used, and the same for the *Values Plot* area, only if this output is chosen for (at least) one of the sensors it will be shown. There are though, some elements that are always built and integrated in the main window. The *Status Information* and the *Sensors List* items have this behaviour, but can, like all the items in window, be hidden if necessary.

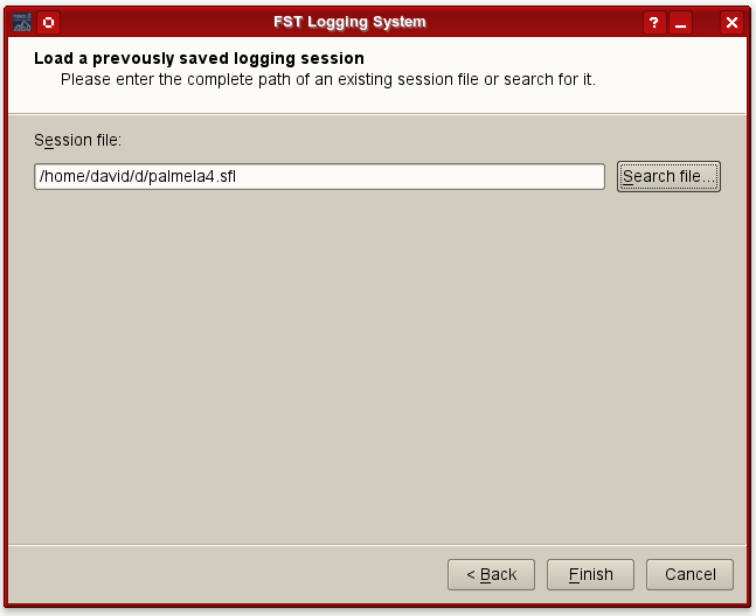


Figure 3.6: Finding a previous session file (*Offline session*).

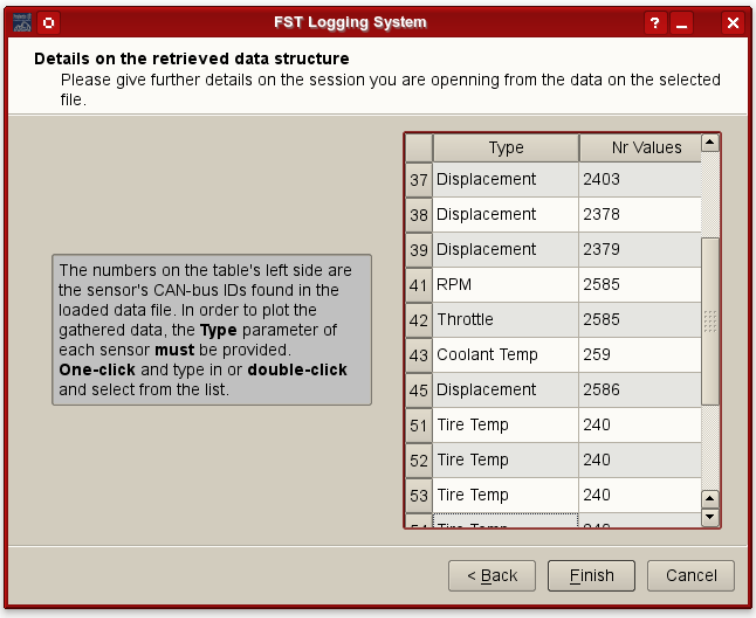


Figure 3.7: Adding the *type* information when data is read from the USB memory's file (*Offline* session).

There is a sensor display widget for each type of sensor in the system for displaying its current value. These are built using classes *dialMeter*, *scaleMeter* and *steeringMeter*. In figure 3.9 the 8 different displays available are shown.

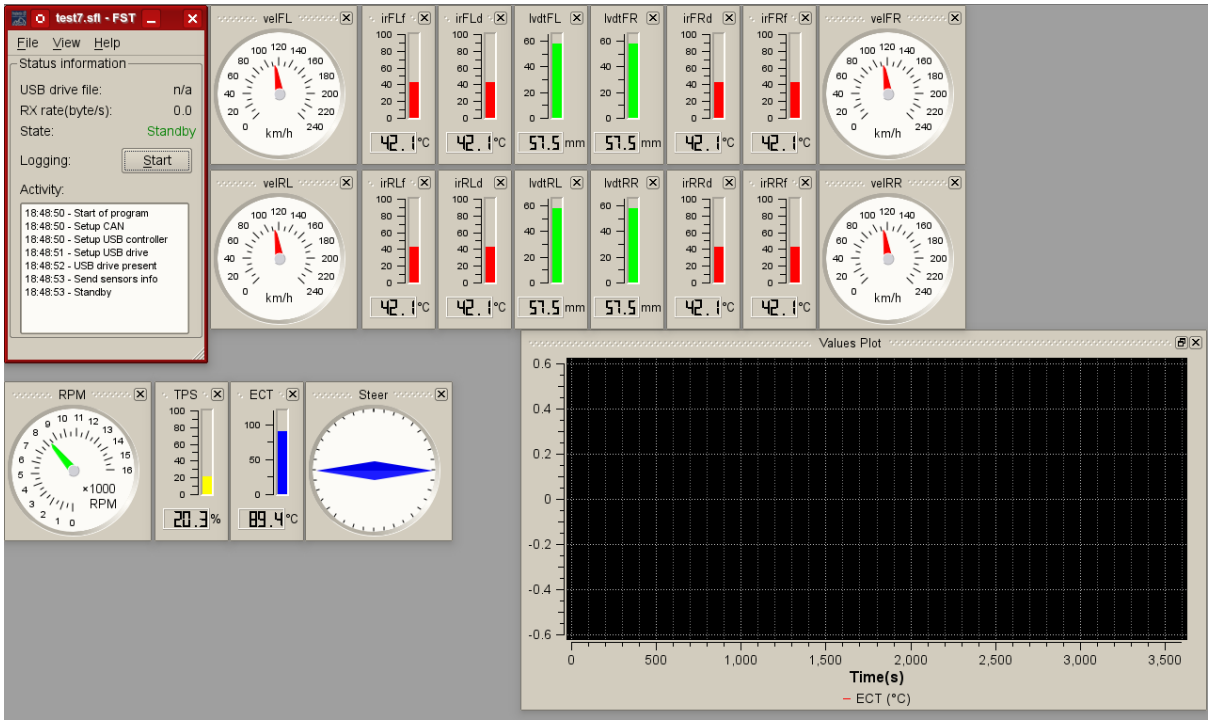


Figure 3.8: Main window in an *Online* session.

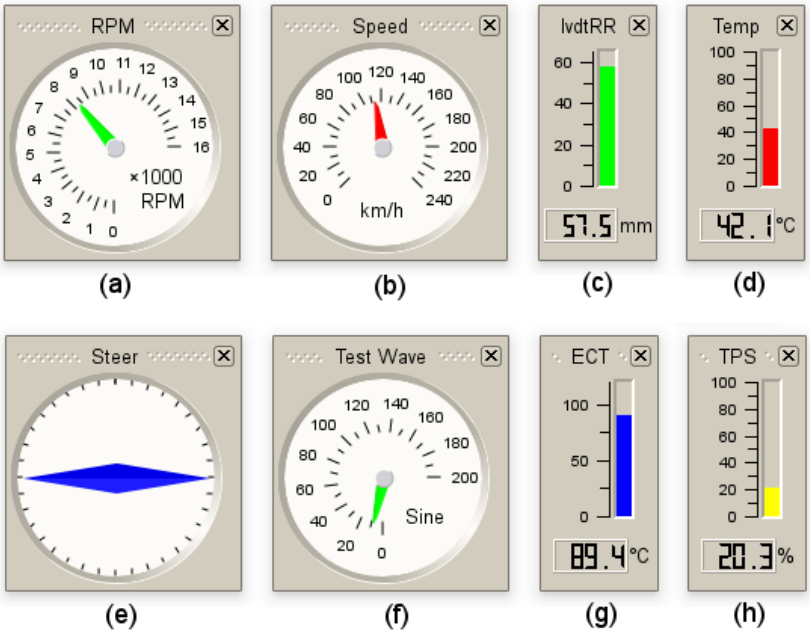


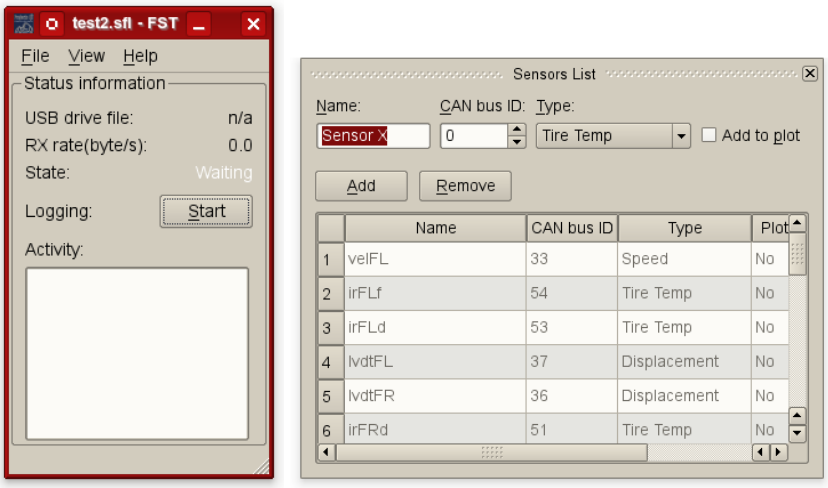
Figure 3.9: The 8 sensor display widgets. (a) Engine RPM, (b) Wheel speed, (c) Suspension displacement, (d) Tire temperature, (e) Steering position, (f) Sine Wave, (g) Engine coolant temperature, (h) Throttle position.

The *Values Plot* item, displays the incoming data for the selected sensors and enables zooming and axis dragging using the mouse buttons, besides showing coordinate value with just a click. The current plot image can be saved as an image file (.jpg, .png or .bmp) by chosing this option on the *File* menu.

The *Status Information* item, that can be seen in figure 3.10a, besides holding the log state control button, from which the user can issue the commands seen in table 2.7, shows the name of the file being writen in the USB memory drive, the current wireless link data rate in kB/s and the current state at which the mobile station is. It also includes an activity log so that the user has feedback of what is "going on" in the embarked subsystem.

Everytime the log "Start" button is pressed a new run begins. Consequently a new file for each sensor to hold the data from the run is created and the *Values Plot* is reset.

If there is a change in the topology of the CAN-bus network during a session, the user can, through the *Sensors List* item (figure 3.10b), make the suitable changes, being able to add or delete sensor displaying widgets (and consecuently adding or deleting them from the database).



(a) *Status Information* item (b) *Sensors List* item

Figure 3.10: The two constant components of the *Online* session main window.

***Offline* session** For both modes of this session type (load from hard drive or load from USB drive) the main window has a static structure. There are only two elements, the *Sensors List* and the *Plot Area*, which make available all the controls over each trace on the plot and show the enabled traces respectively.

The *Sensors List*, presents all the sensors available for data analysis. By selecting one, its options become available: it can be hiden and its color and markers can be changed. Because in the same plot area, different quantities are to be represented using only one ordinate axis, every sensor data set (or curve) can be multiplied or have a constant summed to it.

The large *Plot Area* item uses the same widget as the above mentioned *Plot Area*, it enables the user to have a good resolution when analyzing the data. The current plot image can also be saved as an image file in the same way as described above.

In order to build the window and its items two classes are used, *windowItem* and *mainWindow*. The first, is the class of the sensor visualization items and holds their details, the second has the methods to build every kind of item including the *Status information*, *Sensors List*, and *Values Plot*/*Plot Area*.

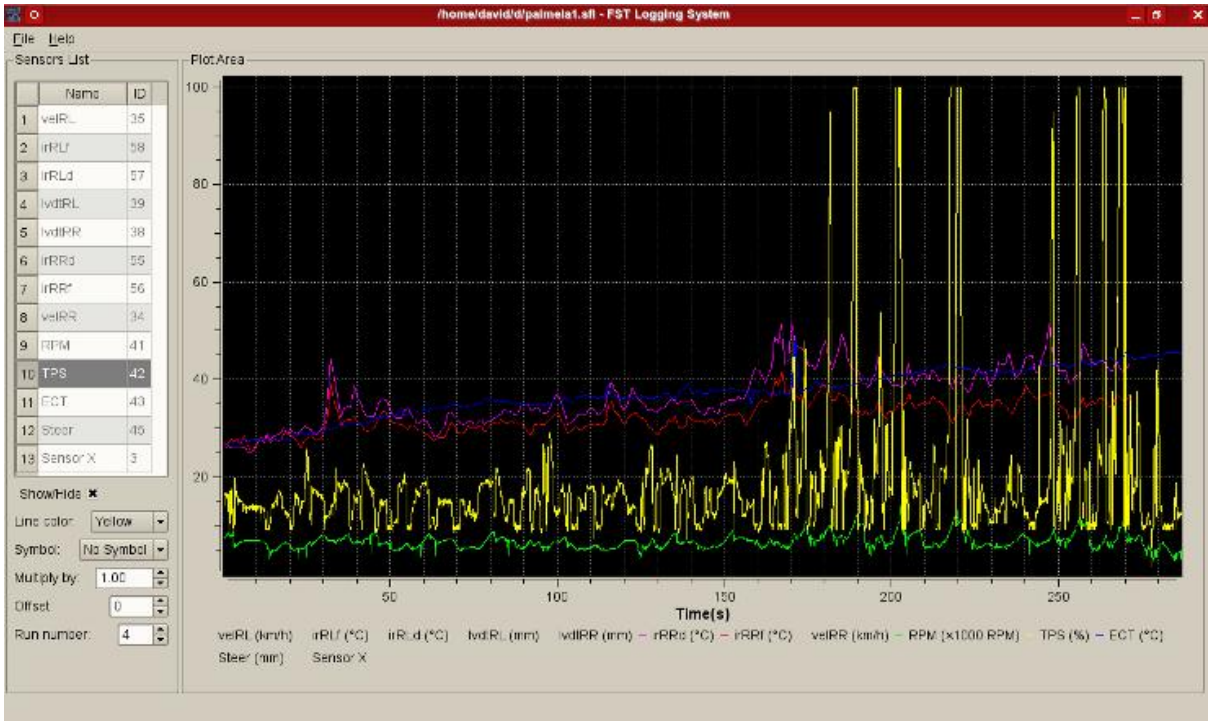


Figure 3.11: Main window in an *Offline* session.

Serial interface

Is the piece responsible for getting the data from the serial port to the right place in the database by interpreting the messages that arrive. The CAN_ID field (section 2.8.7) of the frames sent through wireless is used to evaluate each frame and from there the values are stored in the correct database entry and a signal is sent for the sensor visualization items on the main window to update their values.

The serial interface is implemented through the *serialComms* class. For a better performance the main serial interface method, *run()*, is implemented inside a thread.

Chapter 4

System performance

In order to evaluate the entire system's performance, tests are run in two different environments - the bench and the track. The differences in the two locations for testing are mainly resumed to the conditions of wireless data propagation, size of the CAN-bus network which it connects to and also the conditions in terms of mechanical stress and electrical noise.

4.1 Bench tests

In the bench, under a fully controlled surrounding the system's working condition is thougth as being ideal. Besides being the system's development environment is also where optimal conditions in terms of power supply, wireless distance and mechanical stress are met.

4.1.1 CAN-bus network emulation platform

Since this work has the objective of connecting with a network of nodes supported on a CAN-bus, during its course, a platform for tests was developed so that the interaction with such network could be emulated. This testing platform is composed of simple modules that can be connected through a CAN-bus and generate data in a similar way as the CAN-bus network nodes of the vehicle do.

Each module consists of a CAN-bus communication capable microcontroller (PIC18F4585) fitted with minimal connections and components and a CAN-bus transceiver (MCP2551) as can be seen in the block diagram of figure 4.1.

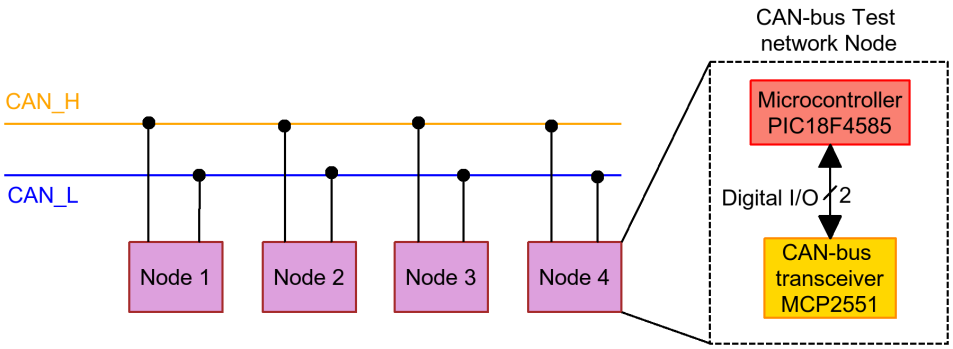


Figure 4.1: Block diagram of the CAN-bus network emulation platform.

The built platform, deppicted in figure 4.2, is formed by 4 of these modules that can be freely connected/disconnected to the CAN-bus. Each node of the testing network is programmed to output (by putting a CAN-bus message on the bus) a sine wave[1] with a certain frequency. By using a sine wave, the data generated in each node is known *a priori* and so it can be easily spoted if the logging of the CAN-bus messages is being successful.

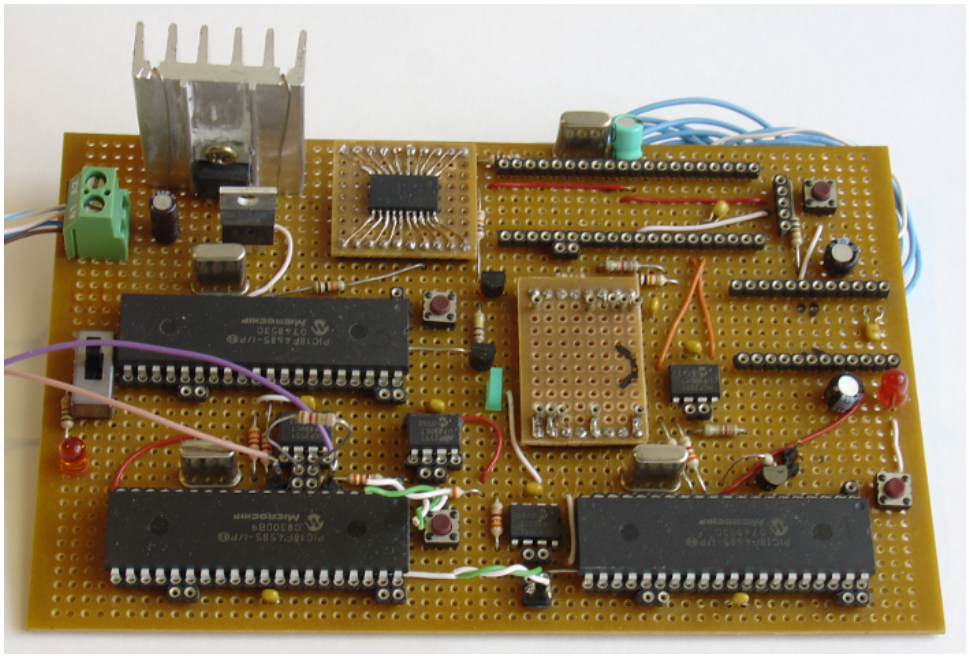


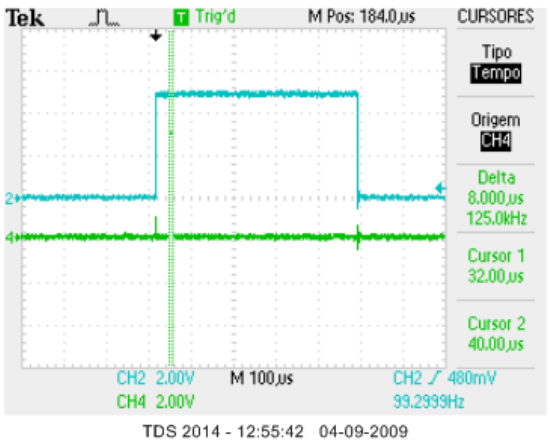
Figure 4.2: CAN-bus network emulation platform built.

[1] Constructed out of a sample of 91 values per quarter of period

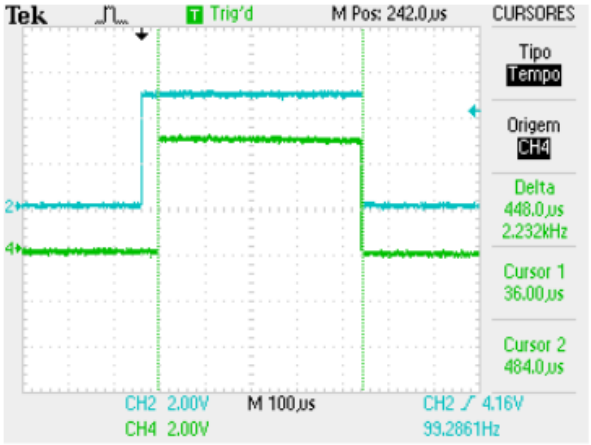
4.1.2 CAN-bus interruption routine timming

As explained in section 2.8.6, each time a new message is put on the CAN-bus, the mobiles station's CAN interruption routine picks it up and sends the CAN-bus messsage data field through the wireless link (using the UART to connect to the radio transceiver). Since the MCU can only deal with one CAN-bus message at a time, the amount of time this process takes is crucial to understand what is the least time spacing that messages put in the CAN-bus can have between them. In order to quantify the time a function takes, a random MCU i/o pin is set high throughout its duration and captured with an oscilloscope. For this test, one node of the CAN-bus network emulation board is connected to the CAN-bus, and it sends messages periocally as explained above.

The two important actions made by this routine are the recording of the data frame on the USB memory drive (call to function vinculumWriteToFile()) and sending it through the serial link to the radio module (call to function sendFrame()). Likewise the time taken by each of these function calls is measured and the results can be seen in figure 4.3.



(a) isrCanrx1() in blue and vinculumWriteToFile() in green



(b) isrCanrx1() in blue and sendFrame() in green

Figure 4.3: Results of the measurements of vinculumWriteToFile() and sendFrame() function calls' time.

From figure 4.3, it is easy to understand that the reason for the overall time, taken by the CAN interruption routine, is the call to function *sendFrame()*, which takes 448 μs, and not the call to *vinculumWriteToFile()* that takes 8 μs. This result is in discordance with the aproximated transmission time computation that can be done based on the baud rate setting of the UART communication channel. As mencioned before (section 2.8.5), the baud rate is set to 115200 baud, which means that the line switches state 115200 times per second. Since each switch in state is equivalent to 1 bit, the time a bit takes to be transmitted is:

$$t_{bit} = \frac{1}{115200} = 8.681\mu s, \tag{4.1}$$

and for a data frame with 7 bytes (section 2.8.7) it makes the time to transmit a message:

$$t_{message} = 7 \times 8\ bit \times t_{bit} = 486.1\mu s, \tag{4.2}$$

which is higher than the measurement made. The difference is found to be due to the MCU baud rate setting. According to [11, page 232], for a device's oscillator frequency of 40 MHz and a UART baud rate setting of 115200 baud, the actual rate has a positive error to the set value and is 125000 baud. If the time per bit and the time per message are computed again:

$$t_{bitREAL} = \frac{1}{125000} = 8\mu s \rightarrow t_{messageREAL} = 7 \times 8\ bit \times t_{bitREAL} = 448\mu s, \tag{4.3}$$

which agrees with the measurement of figure 4.3b.

4.1.3 Wireless link maximum throughput

In order to obtain the efective maximum data transmission rate achieved by the wireless link between the mobile and base stations, the test setup of figure 4.4 is used, where $d = 1\ m$.

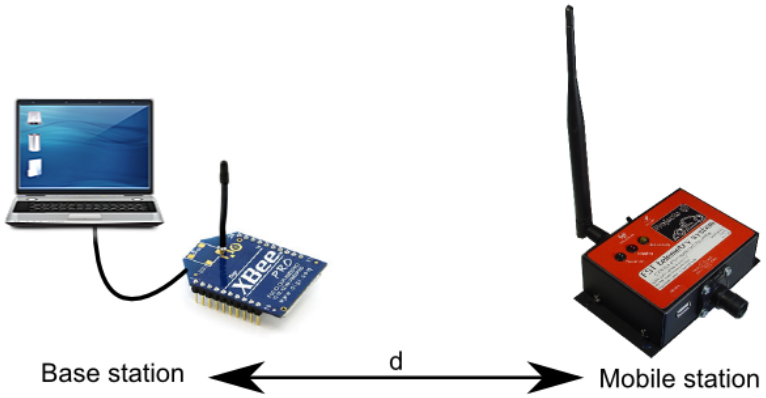


Figure 4.4: Setup for the maximum throughput measurement on the bench.

Since this test has the objective of studying the behaviour of the wireless connection between stations, the mobile station is disconnected from the testing CAN-bus and the main program is changed so that data frames containing the sine wave (used in the same way as explained before) values are sent to the base station at a certain frequency. This way, by changing the frequency at which the frames are sent and computing the amount of bytes arriving per second[2] at the receiving end (base station), the throughput of the wireless link can be obtained as can be seen in figure 4.5.

[2]The throughput value is computed by making an average of the bytes/s arriving in a period of 60 seconds.

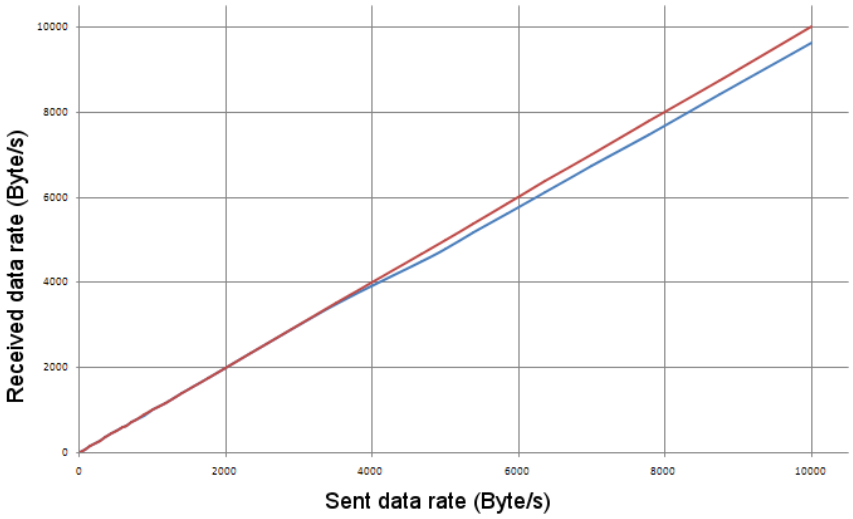


Figure 4.5: Results of the measurements of the wireless link throughput in the bench. Expected value in red and the system's value in blue.

A maximum throughput of 9.61 kB/s is achieved by the wireless link with a relative error to the expected of 0.04.

4.2 Track tests

The system was also tested in a real environment, that is, connected to the vehicle's CAN-bus network of sensors, with the vehicle running on the practice track mencioned before (figure 2.6). The objective was to evaluate the performance of the system in the ambience in which it was designed to work and also to understand if its enrollment in an electrically noisy system would cause failure. Besides the functioning considerations, overall mechanical design and resistance was also evaluated.

The CAN-bus network of sensors was composed of 3 nodes[3] placed throughout the vehicle as can be seen in figure 4.6.

[3]Further details on the CAN network nodes and sensor specifications are considered out of the scope of this work.

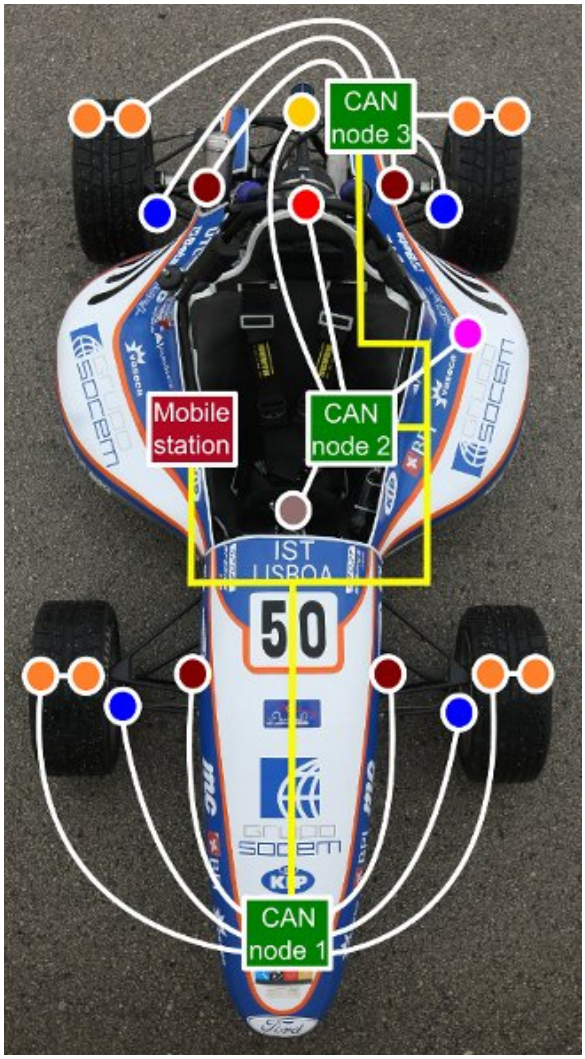


Figure 4.6: Sensors network on the vehicle. The nodes (green) are connected to the mobile station (red) through the CAN-bus (yellow). Sensors: tire temperature (orange), wheel speed (blue), suspension displacement (dark red), throttle (yellow), engine RPM (red), engine coolant temperature (pink) and steering angle (gray).

4.2.1 Performance for different base station positioning

With the vehicle running on the track, the distance between the mobile and the base station is the key factor, for the good operation of the system, being changed with time. In this way, it is of interest to study what is the best location around the course to place the mobile station. Two different locations are used and they are chosen for both being out of the limits of the track and being believed to turn out the best results in terms of wireless propagation, due to having less obstacles or less distance to the track. In figure 4.7, the two locations can be seen. Location 1 (blue square) is on top of a small garages building, approximately 4 meters high, from where all the track can be seen, and Location 2 (red square) is by the course's start line, at approximately 1.5 meters in height.

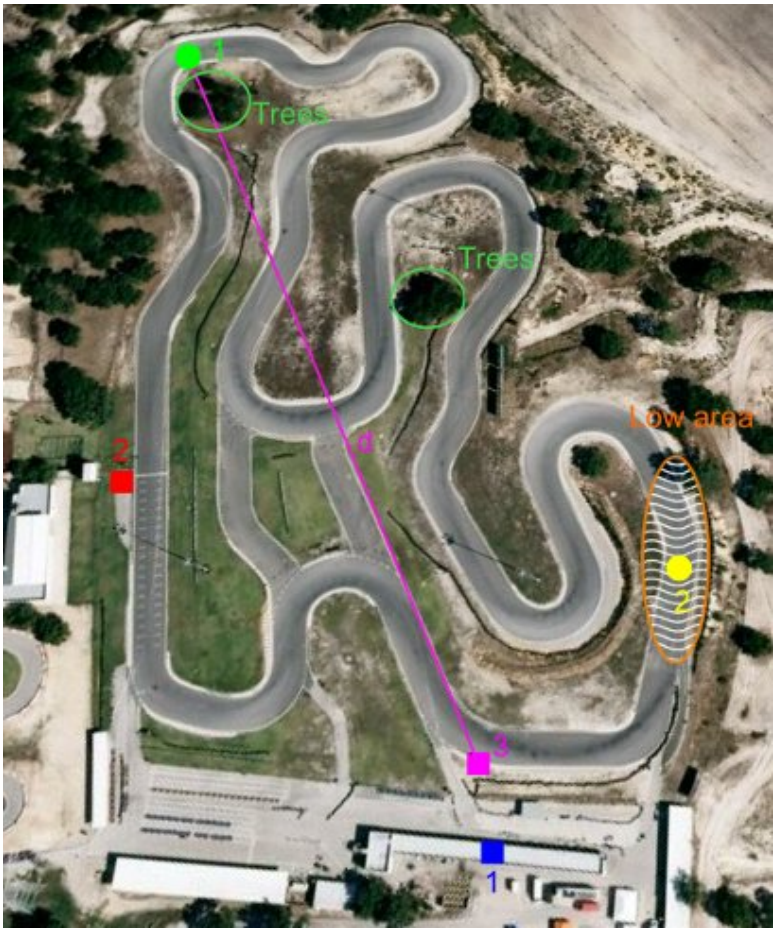


Figure 4.7: Practice track image with points of interest. Location 1 (blue square), Location 2 (red square), Track point 1 (green circle), Track point 2 (yellow circle), Trees (green elipse) and Low area (orange elipse).

In order to evaluate the wireless link, the emulation platform (explained above in section 4.1.1) is added to the CAN-bus network already in the vehicle, with just one node generating a sine wave, in order to, as before, have "known" data being generated on the embarked system. By looking for imperfections on the sine wave shape (figure 4.8, captured with the *Save plot image* function described in section 3.3.3), it was possible to identify the spots on the track where the wireless connection between the mobile and the base station was temporarily broken as the vehicle performs several laps. The "higher" connection loss spots of the track are marked on figure 4.7. In this figure, it can be seen that with the base station at Location 1 the critical Track points are 1 and 3, which is belived to be due to trees in the station's line-of-sight propagation[4], and for Location 2 the critical Track point is number 2, due to the track going through a lower ground area at that point.

Overall, the performance in both locations is similar, as the position of objects in the middle of the track area creates dificulties for both in different spots, and the fact that from one Location to the other, the distance to the vehicle is exchanged by height (which translates in line-of-sight propagation between stations).

Although the method used enabled some weaker connection spots on the practice track to be iden-

[4]Line-of-sight propagation refers to the travelling of radio signals in a straight line with no obstructions.

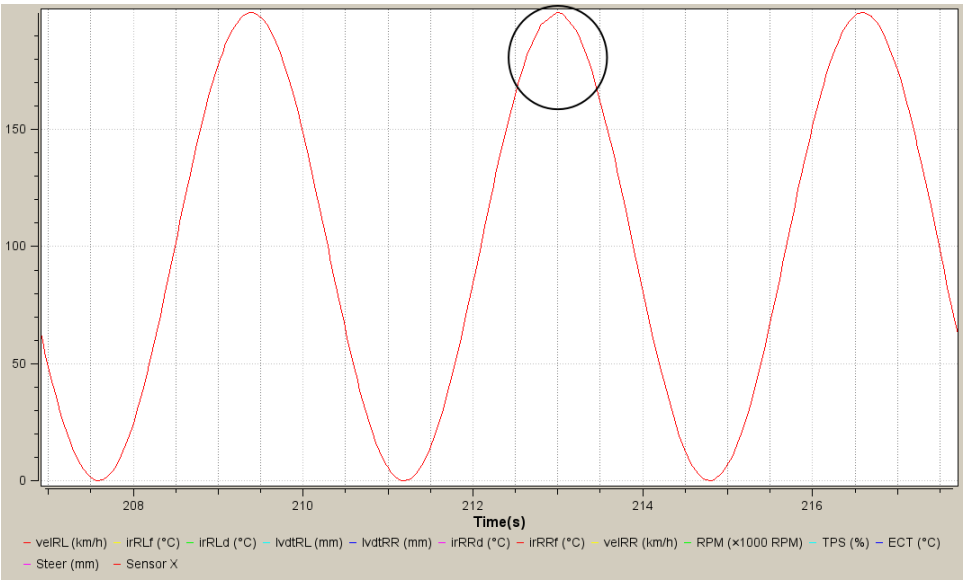


Figure 4.8: Example of a sine wave with short imperfections (black circle).

tified, the peformance/coverage of the connection is good, as, considering the average speed at which the vehicle performs the laps (measured to be always above 40 km/h on this track), the losses in "live" sensor data are acceptable and barely noticed, also due to the frequencies at which data is acquired from each sensor (in the case of the test realized: 10 Hz for all sensors except temperature - 1 Hz). In figure 4.9 a saved image of the plot area shows an example of the data gathered by the system at the practice track.

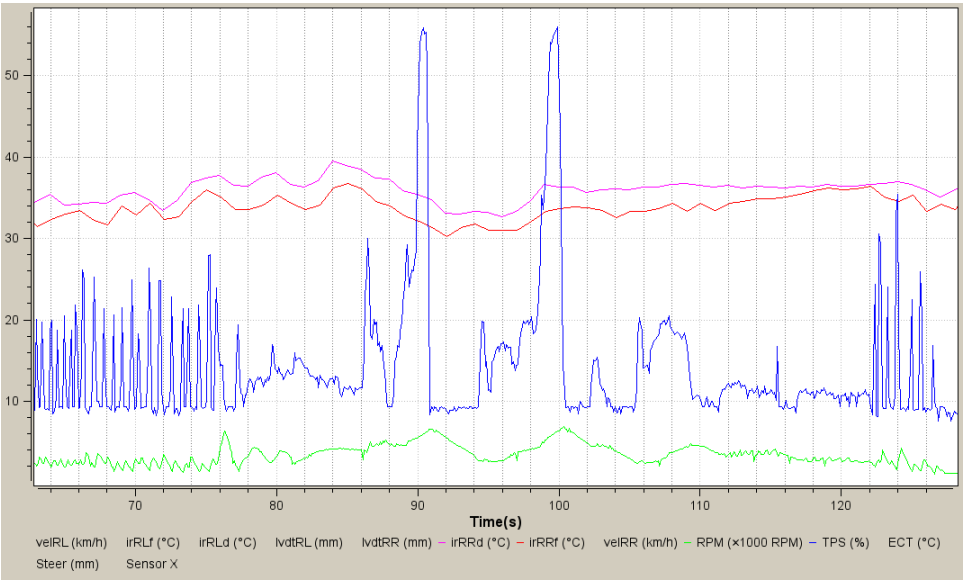


Figure 4.9: Data acquired at the practice track. Temperatures of the tire's inside (pink) and outside (red), the throttle's position (blue) and the engine's RPM (green).

4.2.2 Wireless link maximum throughput

The efective maximum data transmission rate on the track was obtained for what is thought as one of the worst case scenarios, when the car is passing by Track point 1 (figure 4.7) and the base station is at Location 3, which differs from Location 2 on the height, being this approximately 80 cm off the ground. The testing scheme is in all similar to the one of figure 4.4 for the bench test, with the difference being the distance that separates the two stations $d \simeq 300\ m$, as can be seen in figure 4.7. The result of the throughput capability of the wireless link can be seen in figure 4.10.

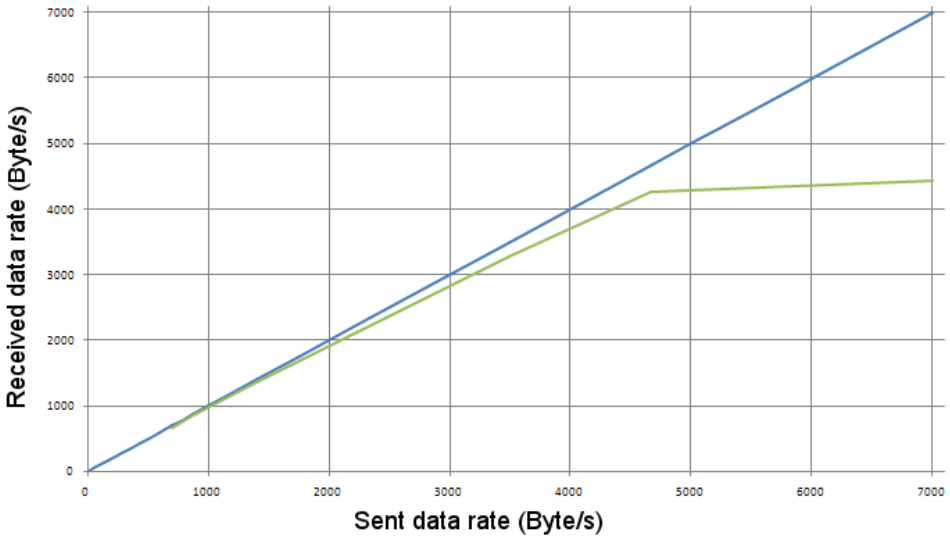


Figure 4.10: Results of the measurements of the wireless link throughput in the track. Expected value in blue and the system's value in green.

The maximum throughput achieved by the system in this case was 4.45 kB/s with a relative error of 0.36 to the expected, and 4.28 kB/s with a 0.08 relative error to the expected. The first value is obtained with a high relative error, meaning a great amount of losses, and so the second value is considered as the maximum with the system in acceptable working conditions.

4.2.3 Considerations on mechanical behaviour

The mobile station enclosure proved to be resistent to the vibrations of the vehicle, and the USB memory drive socket is strong enough to hold the drive without this one falling. The station's bottom fixation piece was also able to safely hold the unit in place just using zip ties as can be seen in figure 4.11.

Figure 4.11: Mobile station installation on the vehicle using zip ties for secure hold.

Chapter 5

Conclusions

In this project, the development, construction and deployment of a CAN-bus data logging and telemetry capable system was proposed. To reach this objective both COTS parts and original circuit design were put together with software developed for two different platforms and the result was tested in a real world cenario. The work was divided in two parts: an embarked sub-system (the mobile station in chapter 2), composed of a microcontroller, a CAN-bus interface and a storage unit and an interface sub-system (the base station in chapter 3), composed of a PC running an application with a graphical user interface (GUI). In order to connect the two sub-systems, a wireless link was studied (section 2.4.2) and used (section 2.4).

Both stations are original contributions of this thesis and the mobile station hardware was developed having in mind the balance between cost, performance and ease of integration, particularly the use of the *Vinculum* USB host controller (section 2.5.2) together with a USB memory drive, proved a good option when compared to more classic approaches like, SD or MMC cards and EEPROMs. In the mobile station's embedded software, the time taken by the critical routines (section 4.1.2) was minimized in order to collect all the messages put in the CAN-bus and, to the extent tested, CAN-bus messages were not lost.

Regarding the base station, its hardware was kept simple in opposition to its software, which grew in complexity because of the project objective of having a dynamic and versatile user interface. The GUI was developed in a Linux OS but using libraries that allow an easy port to other operating systems.

Of note is the fact that this work was developed to interface with other systems being developed at the same time - the CAN-bus network of sensors (section 4.2). This situation, which is not easily found in the academic course, proved very challenging.

The objective, presented in section 1.1, was completly fulfilled by the delivery of a working system with reliable CAN-bus message recording and wireless transmission of data up to 4.28 kB/s (value that corresponds to the situation tested at the practice track used). This throughput proved enough to transmit the data generated by an existing CAN-bus network of 20 sensors with the vehicle running several regular practice laps and reaching an aproximate 300 m maximum distance between both ends of the wireless link.

The wireless link has, none the less, dificulty to comply with obstacles in the transmission path, being line-of-sight propagation the ideal for the *XBee-PRO* radios, which means a track completely free of obstacles. Due to this, and the fact that no flow control or acknowledge schemes are used in the

wireless transmission of data in either direction, live data from the mobile station can be lost as seen in section 4.2.1.

Concerning the data recorded on the USB memory drive, its safety is only compromised if proper closing of files does not takes place, which can (as proved by experience on test runs) happen, when the vehicle, for example, undergows a problem and its power needs to be swiftly cut. The solution would pass by modifying the recording method to make it more robust in the presence of sudden power failures.

5.1 Future work

As future enhancements of this system, it is proposed the further exploitation of the degree of freedom presented by the antenna type. By using a more directional antenna on the base station, propagation problems could be mitigated. Also the use of flow control and the implementation of a minimal handshaking protocol would contribute to less losses on the transmission. Another option would be to move to WiFi technology which would provide a larger bandwidth and also make easier the implementation of a transport layer protocol like TCP. In order to reduce the power consumption of the mobile station a sleep state can be implemented and the telemetry scheme could be changed in order to run in tracks with a lot of obstacles by using the information on the radio signal strength to choose when the mobile station should transmit.

On the base station's software, it is proposed an enhancement of the serial routine, so that it becomes less CPU "intense", the inclusion of a "driver comparison mode" and also the possibility to do further configurations of the mobile station.

5.2 Cost analysis

As mentioned before, the cost of components was one of the parameters taken into account throughout this work, and with special importance in the choice of the COTS devices included in the system. The amount and cost of each component can be found in table 5.1, together with the efectively paid value and supplier's name.

The difference between the unit price (multiplied by the amount) and the efectively spent for each component has to do with added shipping costs, orders with miminum quantities, mistakes in component order, manufacturers samples service, or free components from miscellanious sources. Left off the table, due to the difficulty in computing a cost, are screws, heat shrink tube, solder and wires. Also left out, is the PC, needed to run the base station software but left out as any computer with a USB port can be used.

In order to compute the total cost of the system, 3 different cases are appreciated and the result can be seen in table 5.2. In the cost of the first (and only) prototype built, which has the highest cost, all the money spent is considered. For the case of building a second prototype, the cost drops dramaticaly as all the material that had to be bought in large quantities for the first prototype is available at unit price, and particularly in the case of the *Development Kit Xbee-PRO*, one of these is enough for 2 systems to be built. Lastly, the production case, takes in consideration a large number production (considers unit

Table 5.1: Prototype cost's breakdown.

Component	Amount	Unit price (€)	Spent (€)	Supplier
Aluminium Box RETEX 105x35x75 mm	1	6.9	6.9	Dimofel
Voltage Regulator LM7805 TO-220	1	0.2	0.2	Radipeças
Crystal 10 Mhz	1	0.16	0.16	Radipeças
On/Off Toggle Switch	1	1.16	1.16	Radipeças
5 mm LED Plastic Support	3	0.1	0.3	Radipeças
5 mm Red LED	1	0.07	0.07	Radipeças
5 mm Green, Yellow LED	2	0.08	0.16	Radipeças
Lithium Coin Cell 3 V 200 mA	1	3.56	3.56	Radipeças
MCU PIC18F4685-I/PT TQFP	1	9.58	0	Farnell
RTC DS3232SN SOIC	1	10.42	0	Farnell
Voltage Regulator TC1264-3.3 SOT-223	1	0.8	0	Digikey
Vinculum module VDIP1	1	24.85	56.66	FTDI online
Radio Transceiver Xbee-PRO U.FL	1	27.91	27.91	Farnell
Development Kit Xbee-PRO	1	179.8	179.8	Farnell
CAN transceiver MCP2551	1	1	0	Farnell
Cable u.fl-SMA 75 mm	1	7.9	17.5	RS
Connector 4vias Sureseal w/flange (min 5)	1	3.02	15.1	RS
Pins Sureseal male (min 10)	2	0.31	6.2	RS
Pins Sureseal female (min 10)	2	0.58	11.6	RS
Enclosure Sticker color print	1	2.6	5	Let's Copy
Spray paint mate black	1	3	3	Leroy Merlin
PCB fabrication 2layers	1	25.86	63.25	Beta Layout
Antenna 2.4 GHz SMA	1	11.48	11.48	RS
Heatsink TO-220	1	0.5	0	RS
Stick-on feet (bag of 108)	4	9.25	0	RS
PCB Socket 20ways vertical (min 5)	1	2.14	10.7	RS
SMD resistors and capacitors	22	0.86	17.6	Farnell

price for every component and no samples), which reduces the cost. To have a complete figure of the production case total it should also be added the assembly and testing cost.

For the first prototype and the production cost cases, a great cut could be achieved by not using the *Development Kit Xbee-PRO*, because this is only used to provide the interface boards between the PC and the radio module on the base station. Simpler and cheaper interfaces are available on the market (for example *Sparkfun*'s *XBee Explorer USB* for €17.48) or a custom board could be developed. The cost for all three cases in this scenario can also be seen in table 5.2, which are computed by deducing the *Development Kit Xbee-PRO* cost, adding 2 of the mentioned interface boards and 1 more *Radio Transceiver Xbee-PRO U.FL*.

Table 5.2: Total cost of the system for 3 different cases.

Total cost	Value (€)	Value by cutting the Development kit (€)
First prototype	400.41	283.48
Second prototype	85.2	148.07
Production	298.69	271.66

Bibliography

[1] Telemotive AG. https://www.telemotive.de/4/en/. accessed 2-Dez-2009.

[2] Cosworth. http://www.cosworth.com. accessed 1-Dez-2009.

[3] Robert Bosch GmbH. CAN Specification Version 2.0, 1991.

[4] IEEE. Standard 802.11g™-2003 - LAN/MAN Wireless LANS. Technical report, IEEE, 2003.

[5] IEEE. Standard 802.16™-2004 - Broadband Wireless Metropolitan Area Network. Technical report, IEEE, 2004.

[6] IEEE. Standard 802.15.1™-2005 - Wireless Personal Area Networks. Technical report, IEEE, 2005.

[7] IEEE. Standard 802.15.4™-2006 - Wireless Personal Area Networks. Technical report, IEEE, 2006.

[8] Fairchild Semiconductor International Inc. LM78XX/LM78XXA 3-Terminal 1A Positive Voltage Regulator, 2008.

[9] Microchip Technology Inc. Application Note 713 - Controller Area Network (CAN) Basics, 2002.

[10] Microchip Technology Inc. High-Speed CAN Transceiver MCP2551, 2003.

[11] Microchip Technology Inc. PIC18F2682/2685/4682/4685 Data Sheet, 2007.

[12] KVASER-The CAN Protocol Tour. http://www.kvaser.com/can/intro/index.htm. accessed 9-May-2009.

[13] Lawicel-CANUSB. http://www.canusb.com. accessed 2-Dez-2009.

[14] Future Technology Devices International Ltd. Vinculum Firmware User Manual Version 2.05, 2004.

[15] Future Technology Devices International Ltd. VDIP1 Vinculum VNC1L Prototyping Module Datasheet Version 0.92, 2007.

[16] Future Technology Devices International Ltd. Vinculum VNC1L Embedded USB Host Controller IC Datasheet Version 2.0, 2008.

[17] Stack Ltd. http://www.stackltd.com. accessed 2-Dez-2009.

[18] Inc. Maxim Integrated Products. DS3232 Extremely Accurate I²C RTC with Integrated Crystal and SRAM, 2007.

[19] MaxStream. XBee™/XBee-PRO™ OEM RF Modules Data Sheet, 2006.

[20] Bhavneet Sidhu, Hardeep Singh, and Amit Chhabra. Emerging Wireless Standards - WiFi, ZigBee and WiMAX. *World Academy of Science, Engineering and Technology*, 21, January 2007.

[21] Race Technology. http://www.race-technology.com. accessed 2-Dez-2009.

Appendix A

Formula Student and Projecto FST

In this appendix, a short introduction to the Formula Student competions can be found, as well as some insight on the team Projeto FST and the vehicle FST03.

A.1 Formula Student

Adapted from *What is Formula Student?* at formulastudent.com:

> Started in the United States by SAE in 1981, the Formula SAE programme was considered to be very worthwhile in providing students with excellent learning opportunities and practical skills. In europe the IMechE accepted the management of the European venture in a partnership with SAE. Formula Student (the name given to the event held in England) is different from Formula SAE in that it is designed to be a progressive learning exercise throughout the different classes. However, the same rules are used for both Formula Student and Formula SAE (with some minor changes).
>
> Formula student provides the students with a real-life exercise in design and manufacture and the business elements of automotive engineering. It teaches them all about team working, under pressure and to tight timescales. It demands total commitment, lots of late nights, and many frustrations and challenges along the way, but the result is the development of highly talented young engineers.
>
> For the purpose of the competition, the students are to assume that a manufacturing firm has engaged them to produce a prototype car for evaluation. The intended sales market is the non professional weekend autocross or sprint racer. Therefore, the car must have very high performance in terms of its acceleration, braking, and handling qualities. The car must be low in cost, easy to maintain, and reliable. In addition, the car's marketability is enhanced by other factors such as aesthetics, comfort and use of common parts. The challenge to the team is to design and fabricate a prototype car that best meets these objectives. Each design is compared and judged with other competing designs to determine the best overall car.
>
> The teams have their work judged by Industry specialists and demonstrate the performance of the car. It is not simply the fastest car that wins, students have to balance speed

with safety, reliability, cost and good handling qualities.

A.2 Projecto FST

The team from IST is formed by students of both mechanical and electronics engineering. It has already successfully delivered 3 prototype vehicles, from which 2 are still in the active. This work is aimed at the latest built prototype, the FST03, who's main technical specifications can be found in table A.1.

At the moment the team is in its fourth formation whom is working on the next prototype vehicle - the FST04.

Table A.1: Technical specifications of FST03.

General dimensions	
Overall Length	2827 mm
Overall Width	1450 mm
Overall Height	1214 mm
Wheelbase	1650 mm
Track	1200 mm(front) 1150 mm(back)
Weight with 68 kg driver	131.85 kg(front) 161.15 kg(back)
Frame	
Construction	Steel tube space frame
Material	4130 alloy steel tube
Engine	
Model	2001 Honda CBR 600 F4i
Cylinders	4
Displacement	599 cc
Power	80 hp at 11500 RPM (measured at the wheel)
Exhaust header design	4-2-1 equal length
Suspension	
Type	Double unequal length A-Arm. Pull rod actuated vertically oriented spring and damper.
Transmission	
Type	Torsen Differential FSAE Special
Wheels	
Rims	13" Carbon Fiber with an 7050 Aluminium Center
Tire	AVON 13" Slicks
Body	
Material	Fiberglass

Appendix B

CAN-bus protocol

In this appendix an introductory explanation of the CAN-bus protocol can be found and also the method to compute the CAN-bus controlling hardware register's configuration values.

B.1 Message arbitration

In order to support a non-destructive bitwise arbitration it is first necessary to define the logic states as dominant or recessive and also have each node monitoring the state of the bus. The CAN protocol defines the logic bit 0 as the dominant bit and the logic bit 1 as the recessive bit. A dominant bit always prevails over a recessive bit in what concerns arbitration, which translates in the lowest Message Identifier (fiels used for arbitration) values having the highest priority. In this way if two nodes are trying to transmit a message at the same time, and since they monitor the bus to check if the voltage level they are sending is actually apearing on the bus, when a recessive bit appears in the lowest priority message the monitored state of the bus will be dominant. Here, the node transmitting the lowest priority message loses the arbitration and, as it notices the loss, it stops transmitting. Meanwhile the message sent by the higher priority node wasn't corrupted and its continued until the end. The losing node keeps monitoring the bus waiting for a period of no activity to send its message.

B.2 Error handling

The nodes on a CAN network have the ability to detect fault conditions and change their behaviour accordingly, implementing in this way fault confinement in the bus. There are five error conditions that relate to errors in CRC calculation, failure to acknowledge, bad message format, bit monitoring and bit stuffing. The number and type of error conditions define in which of three states a node is in. These can be error active, with the node being in full working condition while its transmitting and receiving error counters do not reach 128, error passive, being active but with differences in the way it flags errors until its error counters reach 255 from which it goes on the bus-off state and cannot send or receive messages. This way error confinent is achieved and no faulty CAN node is able to "eat" all the network's bandwidth.

B.3 Message format

In figure B.1 the format of a CAN data frame can be found.

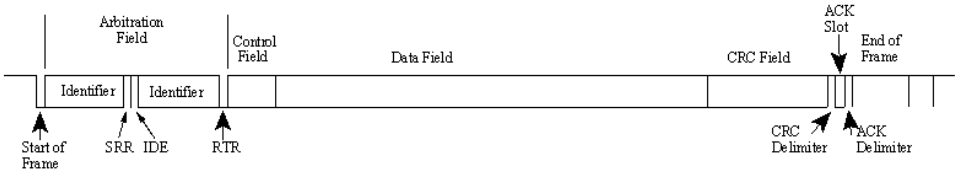


Figure B.1: CAN 2.0B data frame.

B.4 Bit timming

To configure the CAN controller for the network's operating speed proper configuration of the bit timming is needed. Each bit in the CAN-bus is divided 4 segments: the *synchronization* segment, the *propagation* segment, the *phase 1* segment and the *phase 2* segment, as can be seen in figure B.2. The *synchronization* segment is always 1 bit long and exists for clock synchronization. The *propagation* segment allows delays in the bus to be compensated, and the *phase 1 and 2* segments are configured to have the sampling point in the right place.

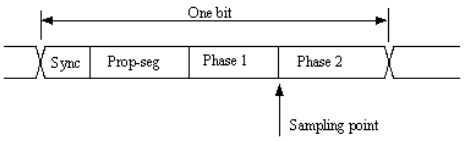


Figure B.2: A CAN bit and its segments.

The time one symbol takes is divided in *time quantas*, and depending on the desired speed for the bus, the number of *time quantas* (TQ) that fit inside a CAN bit varies. This means that in a CAN network the bit rate is the same for all the nodes (the symbol time is the same), what is different, according to the controllers clock (F_{osc}), is the number of *time quantas* that each bit is understood to have. Likewise the registers for each one of the segments are configured accordingly and making sure that the sampling point is between 60% and 70% of the whole bit time.

The remaining registers of table 2.1: *synchronization jump width* and *baudrate prescaler* (BRP) are used for adjustments to the bus clock and as a clock prescaler respectively.

The computation of the register values in table 2.1 starts by defining the speed wanted on the bus, in this case:

$$f_{bit} = 1\ Mbit/s \rightarrow t_{bit} = nTQ = 1\mu s \tag{B.1}$$

and then using the CAN-bus controller equation of [11, page 333]:

$$TQ(\mu s) = \frac{2(BRP+1)}{F_{osc}(MHz)} \tag{B.2}$$

which, having $F_{osc} = 40MHz$ and $BRP = 1$, results in $TQ = 0.1\mu s \rightarrow t_{bit} = 10TQ$.

From here, the *time quantas* are divided by the 4 segments, having in mind where the sampling point is, resulting in the values of the already mencioned table 2.1.

Appendix C

Circuit diagrams of the mobile station board

In this appendix, all the sub-circuit diagrams of the mobile station are shown.

C.1 Microcontroller

The MCU schematic diagram is shown in figure C.1. In order to power the MCU, the 5V supply (C.6) is connected to pins 7 and 28, and GND is connected to pins 6 and 29. Between each pair of power supply pins (Vdd, Vss pins) a decoupling capacitor is used. The MCLR pin (number 18) is active-low, and its held high for normal operation through a resistor to 5V supply. This pin is also the programming voltage input and reset pin. The ICD2 port is used only when the device is being programmed or debugged using Microchip's MPLAB® ICD 2, and left unconnected in normal operation. The LED LlogOn is used as the *Logging* LED in the enclosure (section 2.7.2).

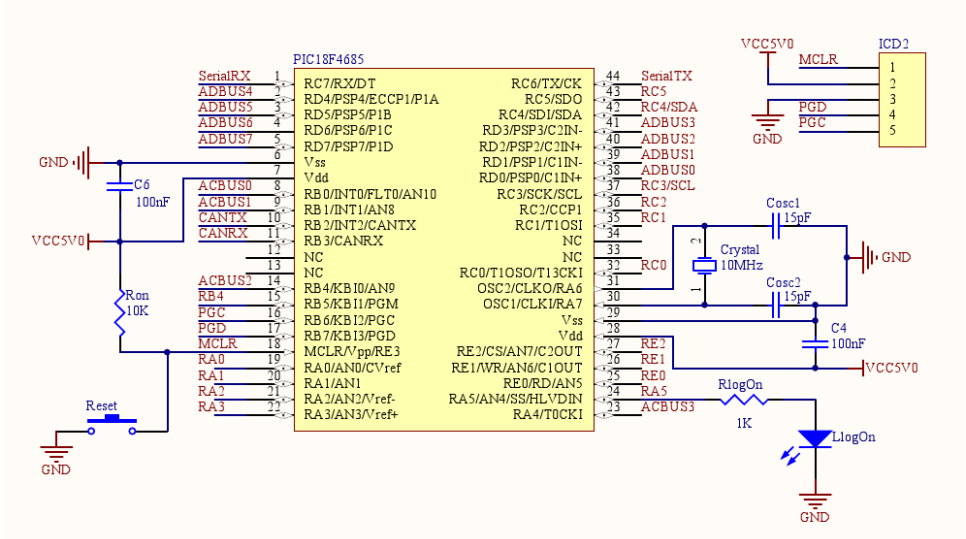


Figure C.1: Schematic diagram of the PIC18F4685.

C.2 Real-Time Clock

The schematic diagram for the RTC is shown in figure C.2. The pin RTCbat in the PortErtcBat is used to connect to battery's positive terminal.

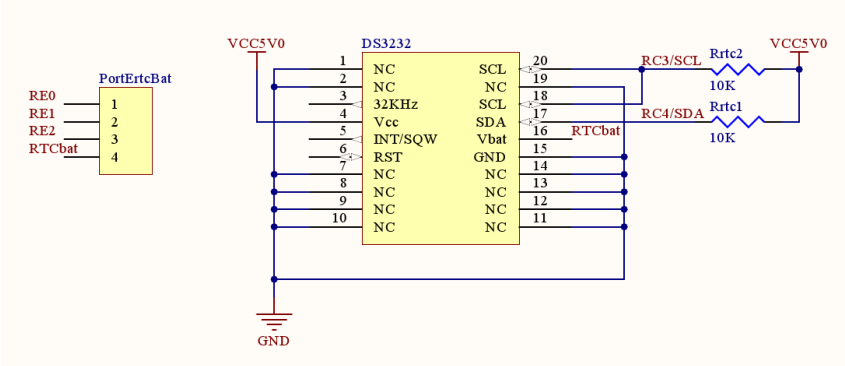


Figure C.2: Schematic diagram of the DS3232.

C.3 CAN-bus transceiver

The schematic diagram for the CAN-bus transceiver is shown in figure C.3. Between the supply pins a 100 nF decoupling capacitor is used.

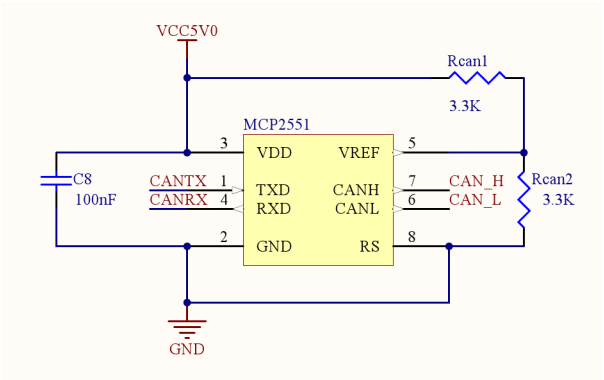


Figure C.3: Schematic diagram of the MCP2551.

C.4 Wireless transceiver

The schematic diagram for the radio transceiver is shown in figure C.4. Between the supply pins a 100 nF decoupling capacitor is used. The LED LxbeeOn is left unconnected in the mobile station.

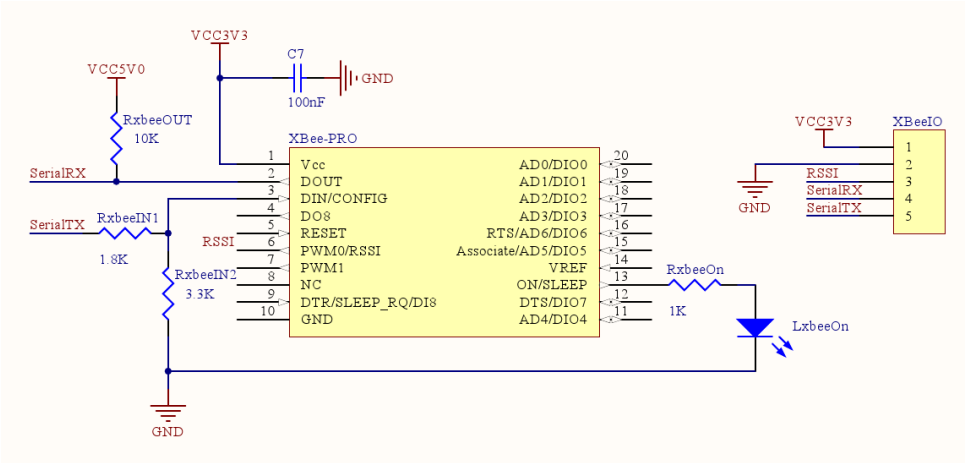


Figure C.4: Schematic diagram of the *XBee-PRO*.

C.5 USB host controller

The schematic diagram for the USB host controller development module is shown in figure C.5. Between the supply pins a 100 nF decoupling capacitor is used. The LED LusbAct is used as the *USB activity* LED in the enclosure (section 2.7.2) and the LED LvdipOn is left unconnected in the mobile station.

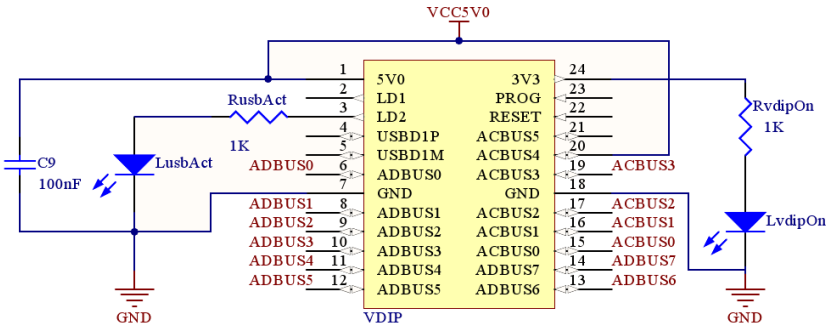


Figure C.5: Schematic diagram of the VDIP.

In table C.1 a description of the VDIP module pins can be found. Note that in this work the bus used is the parallel FIFO (as seen in section 2.5.2).

Table C.1: VDIP1 data and configuration pins for each bus interface.

Pin No.	Name	Pin Name	Type	Description	UART	Parallel FIFO	SPI
6	ADBUS0	AD0	I/O	5V safe bidirectional line, AD bit 0	TXD	D0	SCLK
8	ADBUS1	AD1	I/O	5V safe bidirectional line, AD bit 1	RXD	D1	SDI
9	ADBUS2	AD2	I/O	5V safe bidirectional line, AD bit 2	RTS#	D2	SDO
10	ADBUS3	AD3	I/O	5V safe bidirectional line, AD bit 3	CTS#	D3	CS
11	ADBUS4	AD4	I/O	5V safe bidirectional line, AD bit 4	DTR#	D4	
12	ADBUS5	AD5	I/O	5V safe bidirectional line, AD bit 5	DSR#	D5	
13	ADBUS6	AD6	I/O	5V safe bidirectional line, AD bit 6	DCD#	D6	
14	ADBUS7	AD7	I/O	5V safe bidirectional line, AD bit 7	RI#	D7	
15	ACBUS0	AC0	I/O	5V safe bidirectional line, AC bit 0	TXDEN#	RXF#	
16	ACBUS1	AC1	I/O	5V safe bidirectional line, AC bit 1		TXE#	
17	ACBUS2	AC2	I/O	5V safe bidirectional line, AC bit 2		RD#	
19	ACBUS3	AC3	I/O	5V safe bidirectional line, AC bit 3		WR	
20	ACBUS4	AC4	I/O	5V safe bidirectional line, AC bit 4			

C.6 Power Supply

The schematic diagram for the power supply generating components is shown in figure C.6.

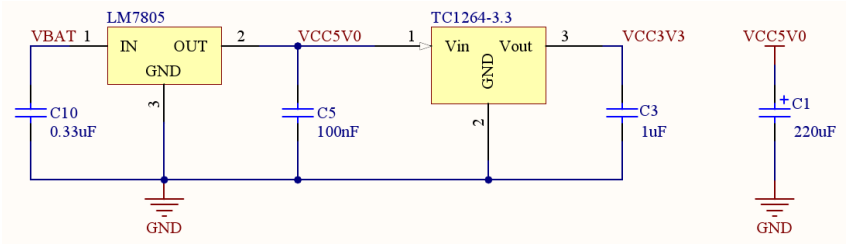


Figure C.6: Schematic diagram of the voltage regulators that power the system.

Appendix D

Radio link

In this appendix the configuration of the wireless transceivers can be found as well as the comparison chart used in the choice of the transceiver modules.

D.1 *X-CTU* software

The vendor's software *X-CTU*, allows the testing of a connection between two *XBee-PRO* radio modules, as well as, configure the devices. Before the tranceivers used in this work are put in their respective stations, they undergow a configuration phase by being connected to a PC running the *X-CTU* software and the core options being set to the values presented in table D.1.

Table D.1: *XBee-PRO configuration options in* X-CTU.

Option	**Value**
Channel	C
ID	3332
Destination address High	0
Destination address Low	0
Source address	0
XBee retries	0
MAC mode	802.15.4 NO ACKS
Coordinator enable	END DEVICE
AES encription enable	DISABLE
Power level	HIGHEST
CCA Threshold	2C
Sleep mode	NO SLEEP
Cyclic sleep period	0
Interface data rate	115200

D.2 Tranceivers comparison chart

Table D.2: Transceivers' details chart.

Name	Frequency(MHz)	TX/RX(kb/s)	Interface(kb/s)	Power(mW)	Sensitivity(dBm)	Range(m)	Modulation	I_{TX}(mA)	I_{RX}(mA)	Weigth(g)	Price(€)
SparkFun LN96	915	9.6	n/a	500	-112	1500	GFSK	450	50	n/a	68.76
Rfsolutions X8200	900	115.2	n/a	500	n/a	2000	F3D,F1D	340	90	n/a	563.58
Rfsolutions 232C-868F	868	38.4	n/a	5	-101	800	FSK	33	33	n/a	143.24
Rfsolutions TinyOne Pro	868	38.4	n/a	500	-105	4000	GFSK	550	35	n/a	113.5
Maxstream XTend OEM	900	115.2	115.2	1000	-100	11000	FSK	730	80	18	138.76
Maxstream XTend Modem	900	115.2	115.2	1000	-100	11000	FSK	480	75	200	231.78
Maxstream XBee-PRO OEM	2400	115.2	115.2	60	-100	1600	ZigBee-QPSK	139	55	n/a	27.91
Maxstream XBee-PRO Modem	2400	250	115.2	60	-100	1600	ZigBee-QPSK	300	90	150	84
Aerocomm AC4424-200 OEM	2400	576	115.2	200	-90	3000	ZigBee	235	85	20	98
Aerocomm AC4790-1000 OEM	900	76.8	115.2	1000	-100	32000	FHSS-FSK	1300	30	20	77.5

Appendix E

Flow diagrams of the mobile station's software

In this appendix the flow diagrams for all the mobile station's functions can be found.

E.1 Real-Time Clock setup and access routines

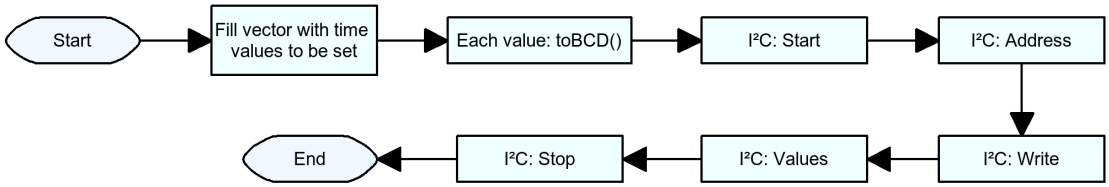


Figure E.1: Flowchart of the setTime() function.

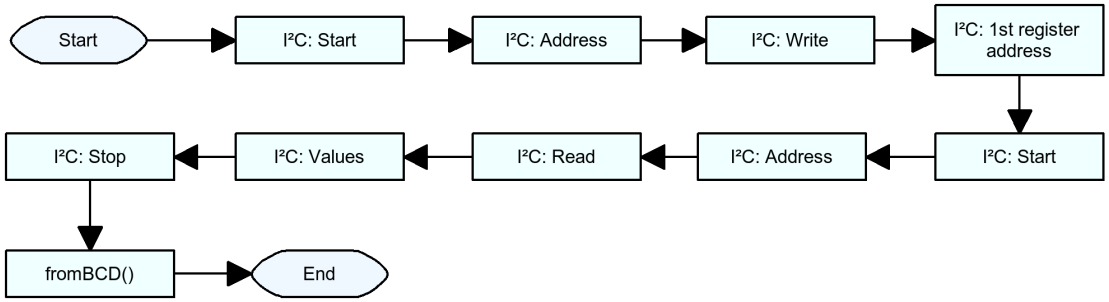


Figure E.2: Flowchart of the readTime() function.

E.2 Main function

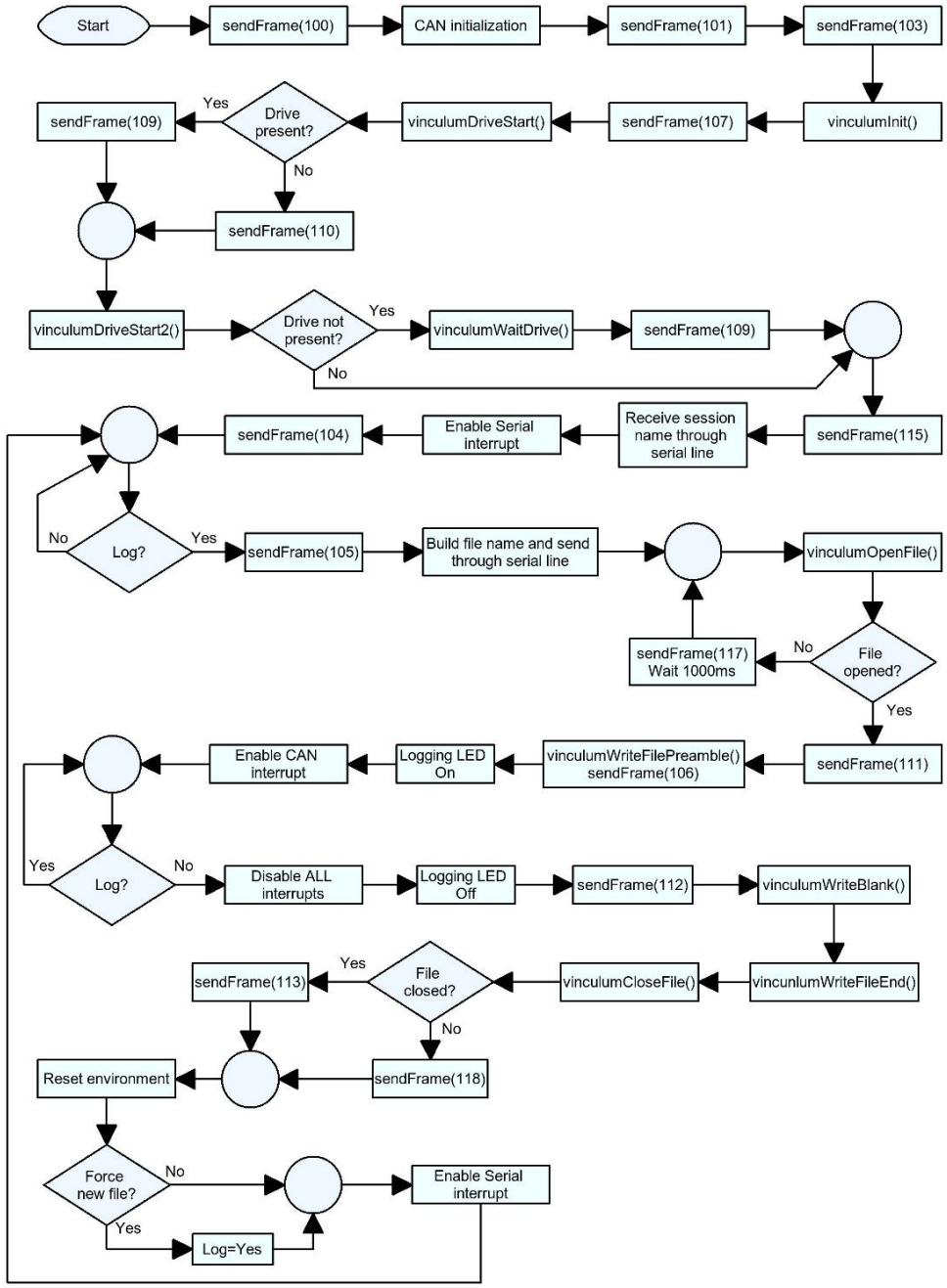


Figure E.3: Flowchart of the main() function.

E.3 Data storage routines

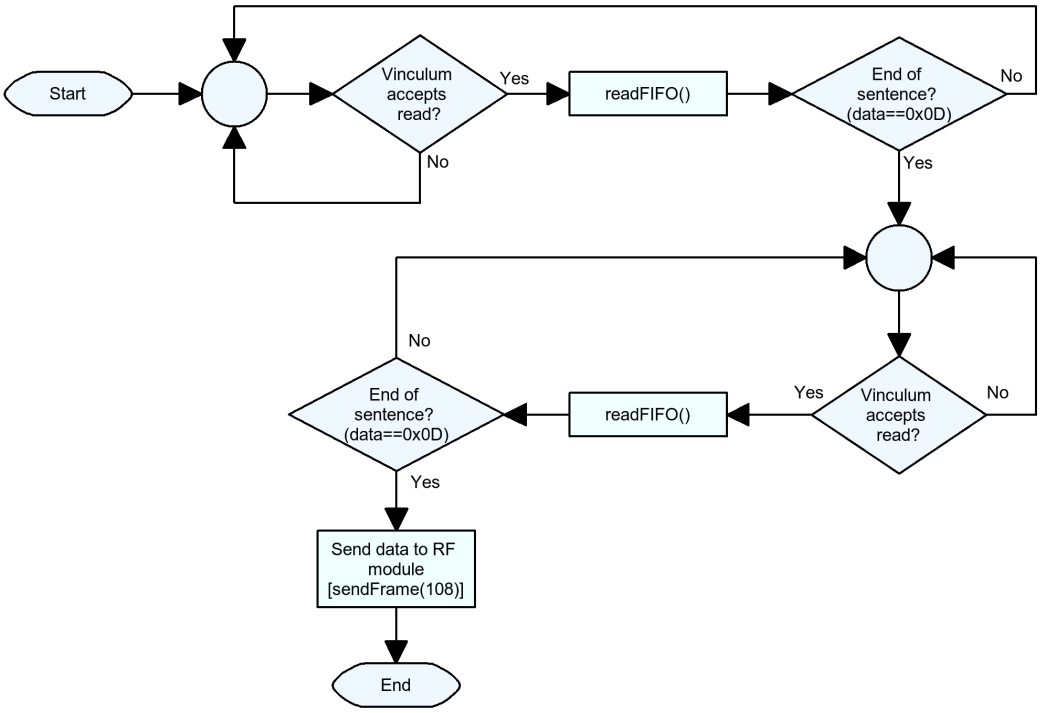


Figure E.4: Flowchart of the vinculumInit() function.

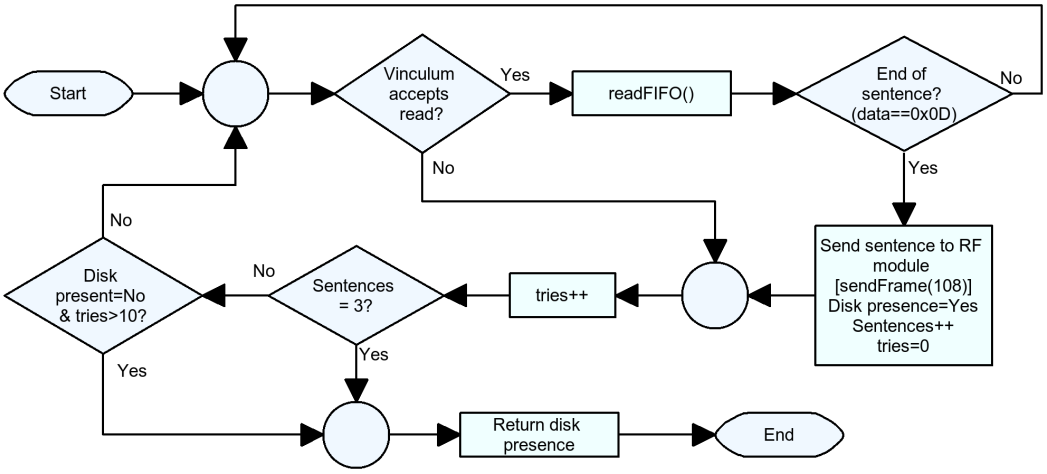


Figure E.5: Flowchart of the vinculumDriveStart() function.

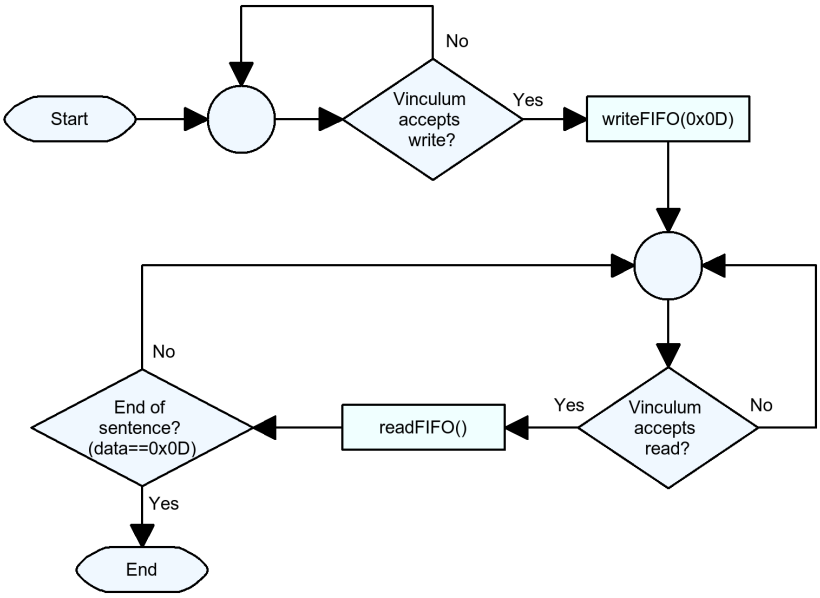


Figure E.6: Flowchart of the vinculumDriveStart2() function.

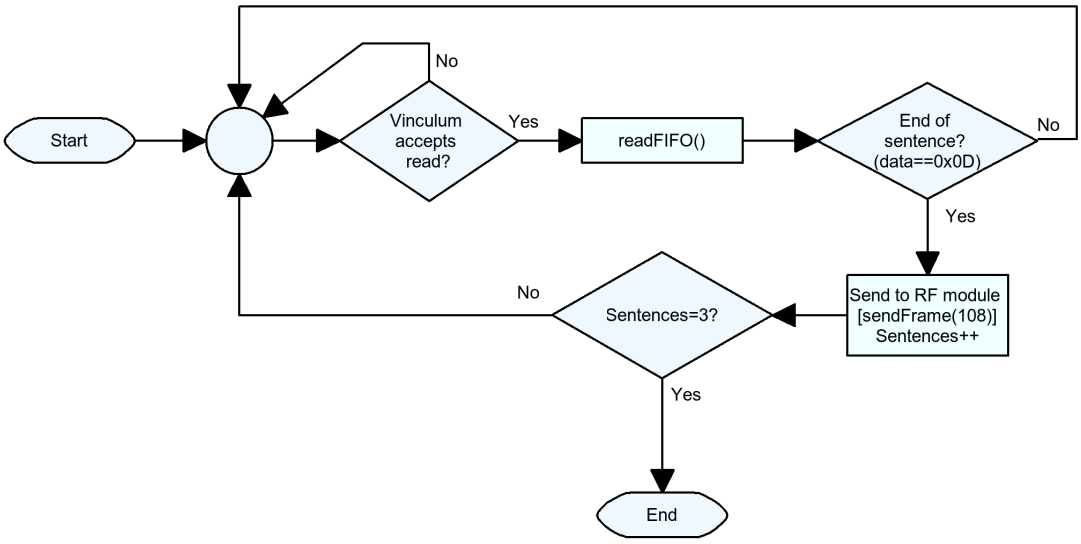


Figure E.7: Flowchart of the vinculumWaitDrive() function.

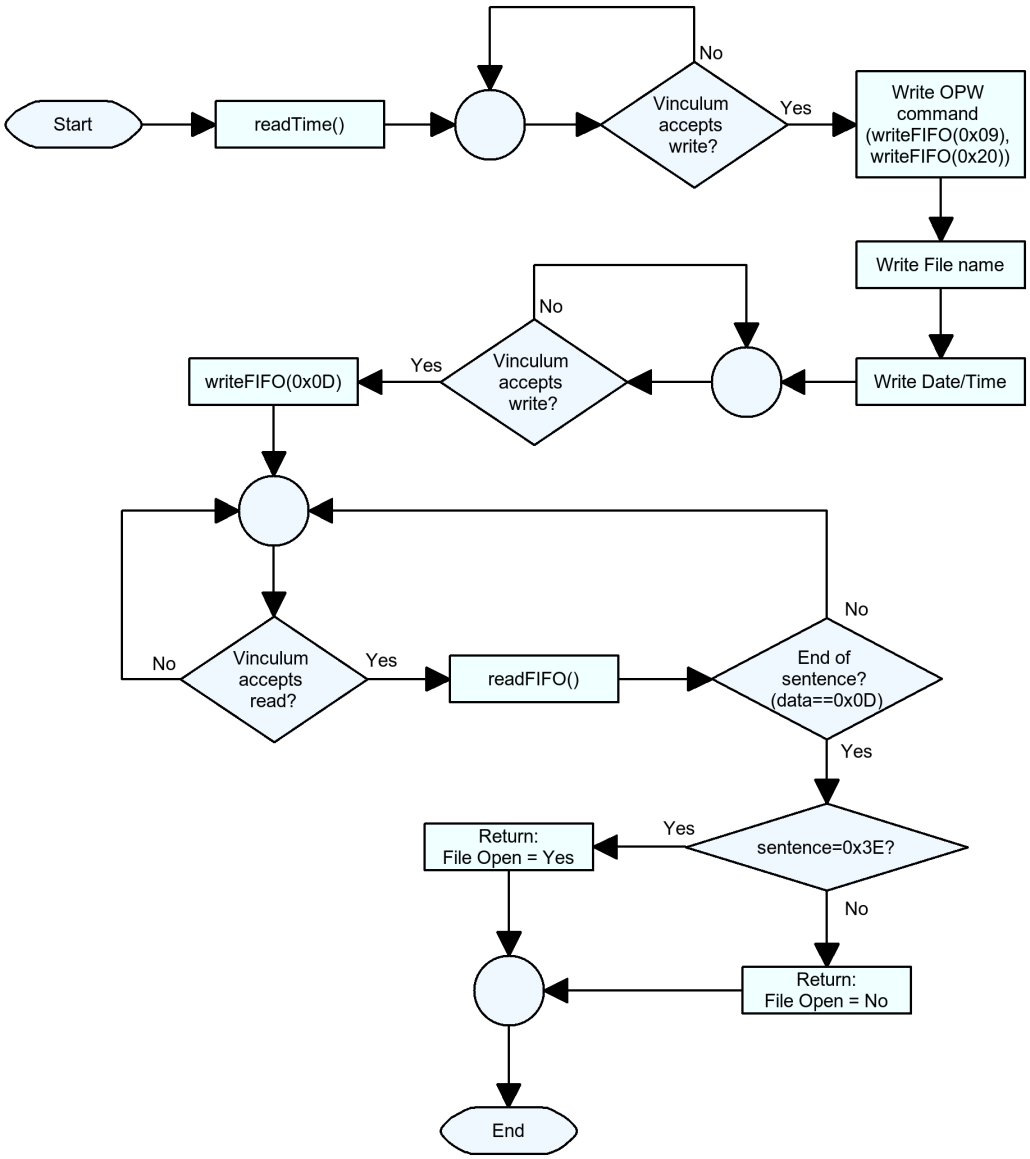


Figure E.8: Flowchart of the vinculumOpenFile() function.

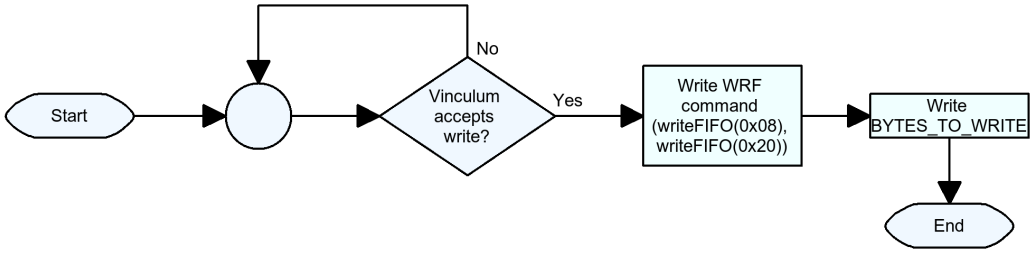


Figure E.9: Flowchart of the vinculumWriteFilePreamble() function.

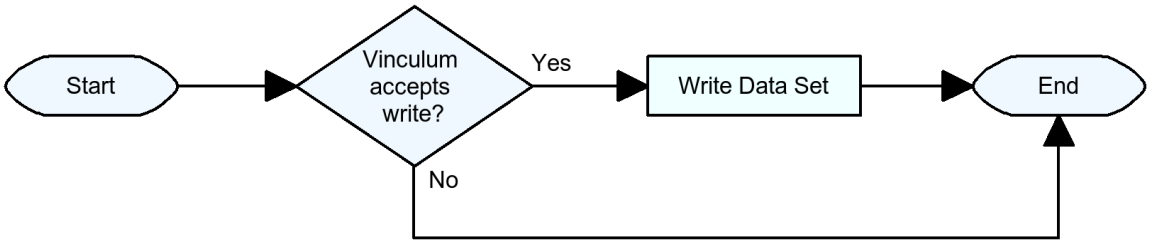


Figure E.10: Flowchart of the vinculumWriteFile() function.

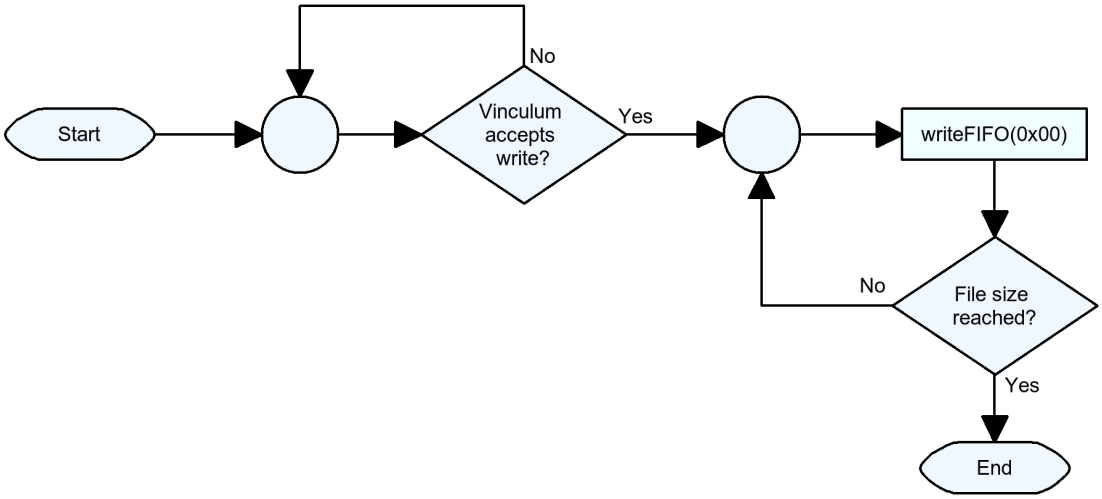


Figure E.11: Flowchart of the vinculumWriteBlank() function.

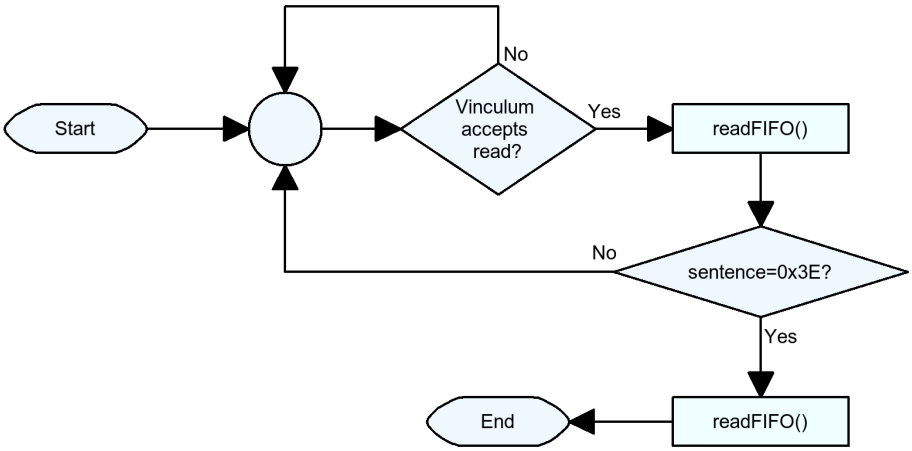


Figure E.12: Flowchart of the vinculumWriteFileEnd() function.

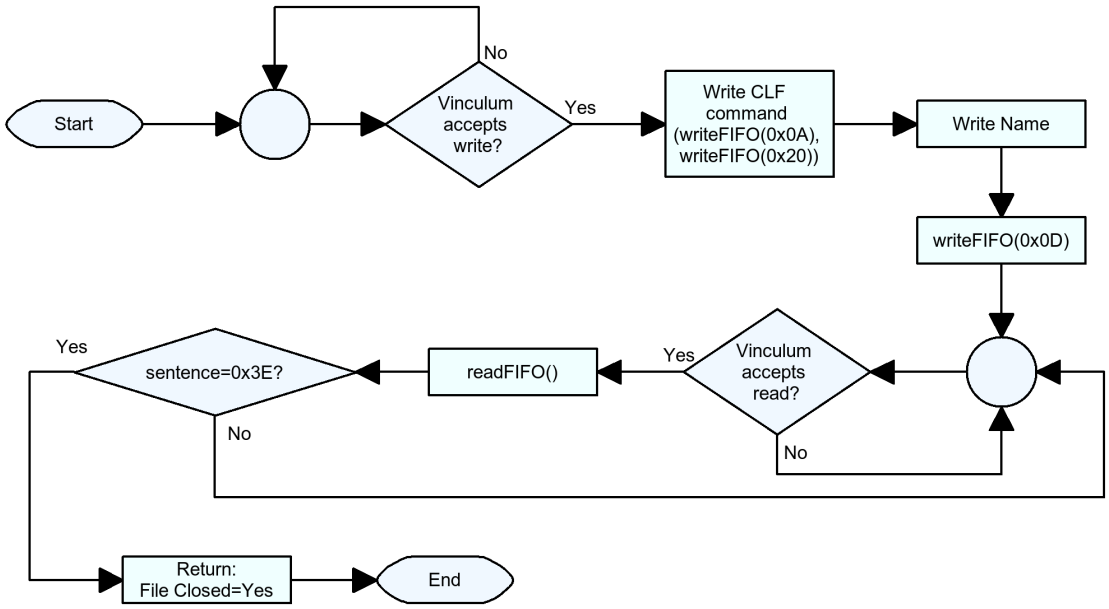


Figure E.13: Flowchart of the vinculumCloseFile() function.

E.4 *Xbee-PRO*/Serial interruption execution

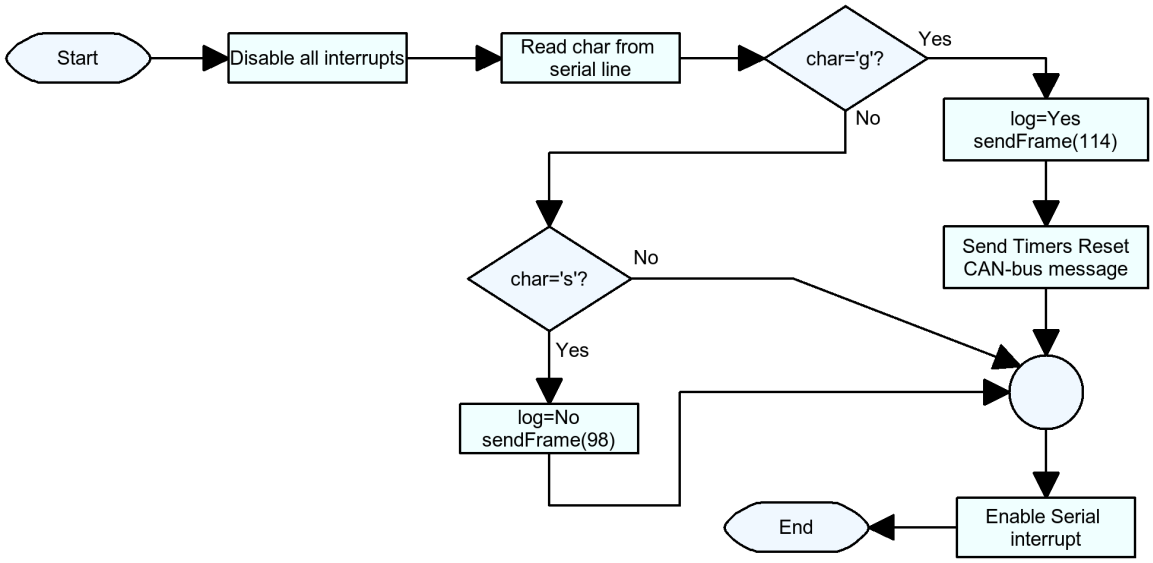


Figure E.14: Flowchart of the isrSerial() function.

E.5 CAN-bus interruption execution

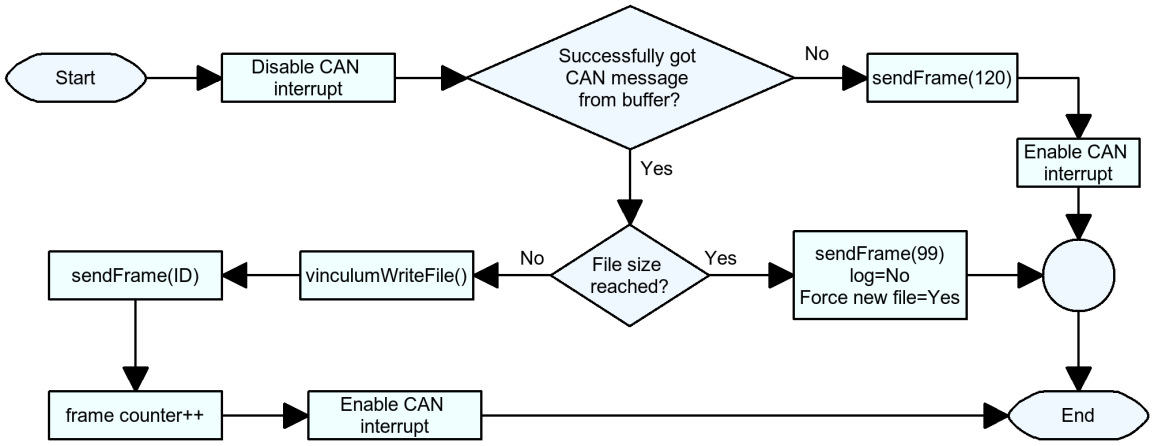


Figure E.15: Flowchart of the isrCanrx1() function.

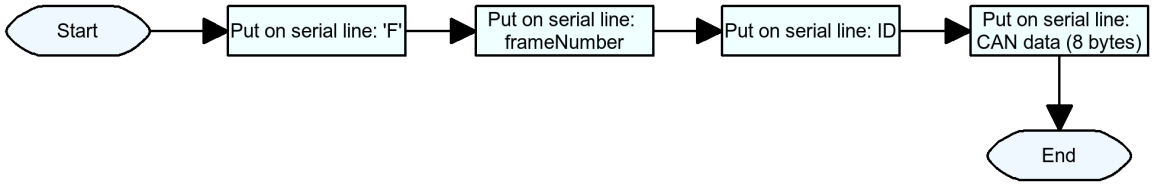


Figure E.16: Flowchart of the sendFrame() function.

Appendix F

Layout of the mobile station's board

The details on the board's layout are gathered in table F.1 and in figures F.1 and F.2 the resulting top and bottom drawings are deppicted. A 3D aspect of the finalized board with components in place is shown in figure F.3.

Table F.1: *Altium Designer* layout details.

Item	**Value**
Number of layers	2
Regular track width	0.2 mm
Battery voltage track width	0.4 mm
Via outer diammeter	1 mm
Via hole diammeter	0.6 mm
Via clearance	0.2 mm
Polygon clearance	0.508 mm
Screws hole diammeter	3.048 mm

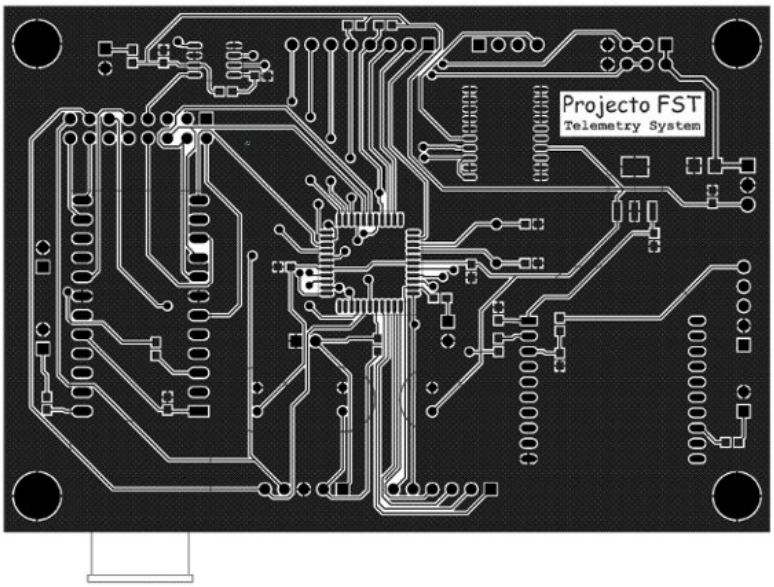


Figure F.1: Top layer of mobile station's board.

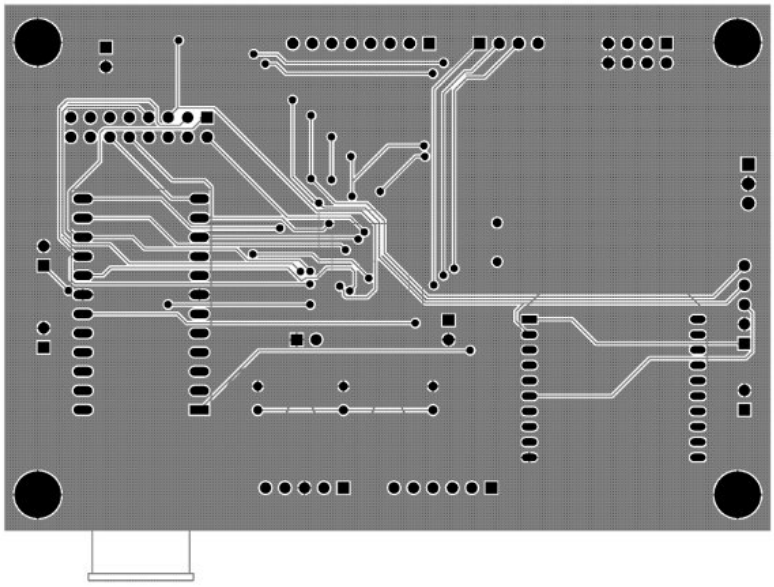


Figure F.2: Bottom layer of mobile station's board.

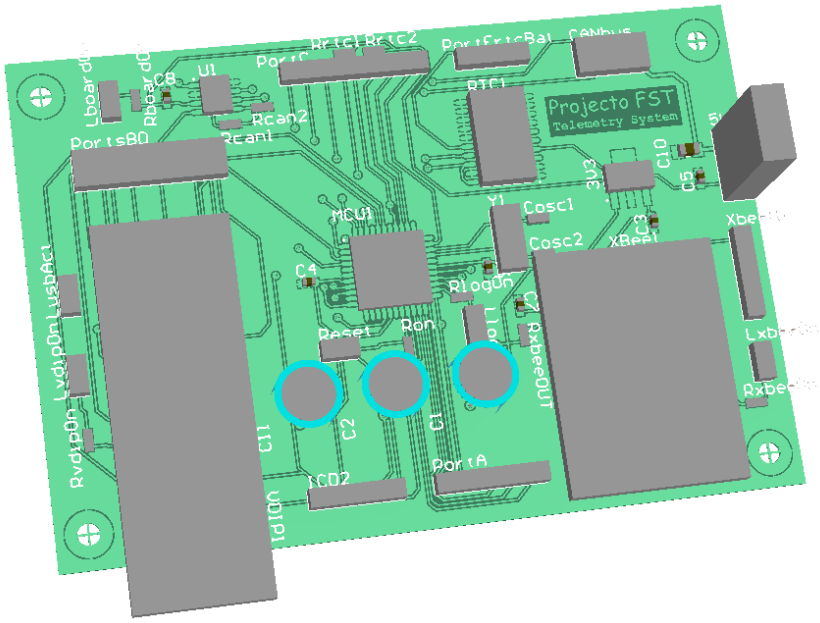


Figure F.3: 3D image of the mobile station's board with components in place.

Appendix G

Base station's software

In this appendix the class diagram of the base station's software is presented and 2 examples of the files created by the program are shown.

G.1 Class diagram

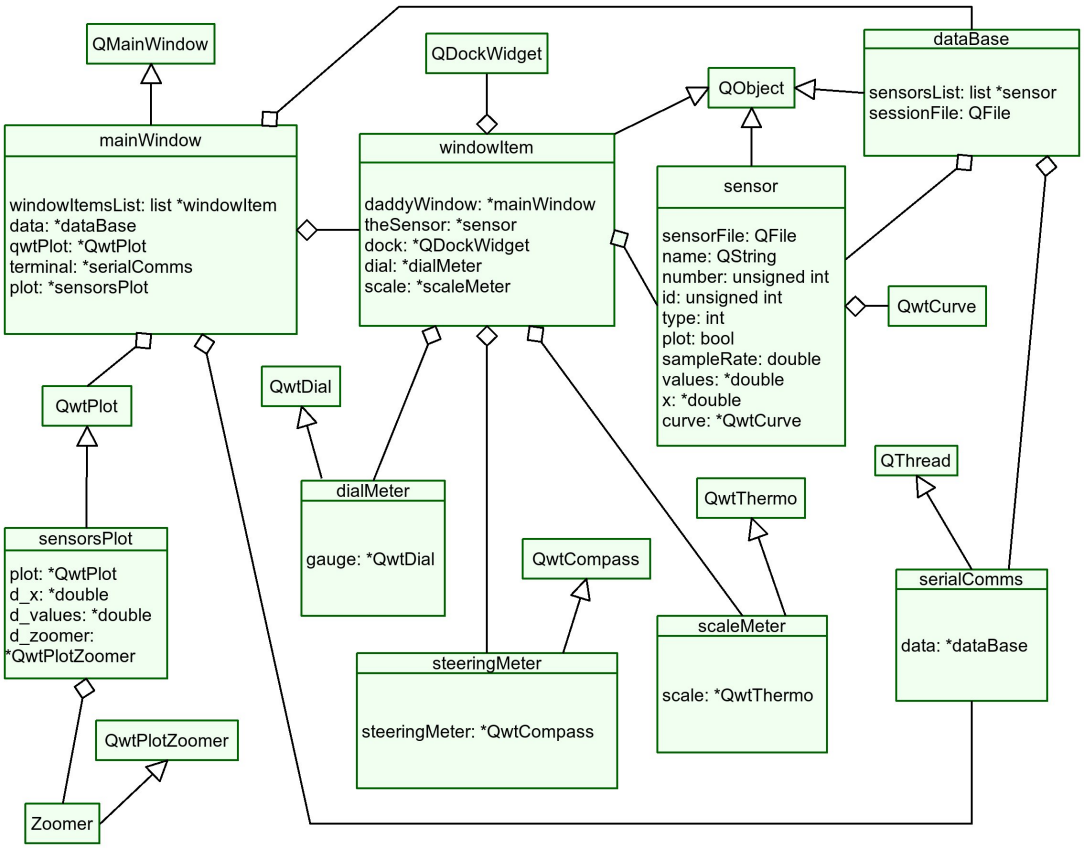


Figure G.1: Class diagram for the base staion's software.

G.2 File examples

Example 1 A session file: test.sfl

```
.FST Logging System - Session file :: Tue 25 August 2009 > 11:32:28
.Files:
test_sensor-1_run-1.txt
2840
test_sensor-2_run-1.txt
285
test_sensor-1_run-2.txt
4157
test_sensor-2_run-2.txt
417
.Number of Sensors:
2
.Number of Runs:
2
```

Example 2 A sensor file: test_sensor-1_run-1.txt

```
.FST Logging System - Sensor file number 1 of Session test.sfl : Tue 25 August 2009 > 11:32:28
.Properties:
.Name:SpeedFL
.Number:1
.ID:33
.Type:3
.Plot:0
.Values:
0.23 0.332639
0.33 0.542346
(...)
```