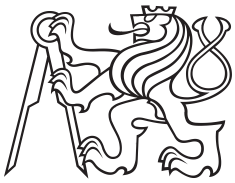
Bachelor Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Electric Drives and Traction

Modular apparatus for arbitrary system datalogging and telemetry

Stanislav Tomášek

Supervisor: Ing. Vít Hlinovský, CSc.
Field of study: Electrical Engineering, Power Engineering and Management
Subfield: Electrical Engineering and Management
May 2017

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Tomášek** Jméno: **Stanislav** Osobní číslo: **420984**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra ekonomiky, manažerství a humanitních věd**

Studijní program: **Elektrotechnika, energetika a management**

Studijní obor: **Elektrotechnika a management**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Modulární zařízení pro telemetrii a sběr dat z libovolných systémů.

Název bakalářské práce anglicky:

Modular apparatus for arbitrary system datalogging and telemetry

Pokyny pro vypracování:

1. proveďte rešerši existujících řešení v technické praxi
2. navrhněte optimální řešení pro sběr dat a jejich následný přenos
3. udělejte návrh DPS (desky plošných spojů)
4. porovnejte náklady na výrobu vašeho řešení s existujícími komerčními produkty

Seznam doporučené literatury:

1. National Instrumens ? NI USB-6008 Multifunction I/O 12bit, 10kS/s - USA
2. Vít Záhlava - Návrh a konstrukce desek plošných spojů - Principy a pravidla praktického návrhu
3. BEN - technická literatura, Praha 2011

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Vít Hlinovský CSc., katedra elektrických pohonů a trakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **09.02.2017** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **27.05.2018**

Podpis vedoucí(ho) práce	Podpis vedoucí(ho) ústavu/katedry	Podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ Datum převzetí zadání

_____ Podpis studenta

BACHELOR THESIS ASSIGNMENT

First name: Stanislav

Last name: Tomášek

Faculty: Faculty of Electrical Engineering, Czech Technical University in Prague

Study programme: Electrical Engineering, Power Engineering and Management

Specialisation: Electrical Engineering and Management

Thesis title: Modular apparatus for arbitrary system datalogging and telemetry

Supervisor: Ing. Vít Hlinovský, CSc., Dept. of Electric Drives and Traction, FEE CTU

Thesis objectives:

1. Research of existing solutions in the industry
2. Optimal solution design for data collection and subsequent transfer
3. Printed circuit board (PCB) design of the device
4. Manufacture cost analysis and cost comparison against other devices

Literature and sources:

- National Instrumens – NI USB-6008 Multifunction I/O 12bit, 10kS/s – USA
- Vít Záhlava – Návrh a konstrukce desek plošných spojů - Principy a pravidla praktického návrhu, BEN - technická literatura, Praha 2011

Acknowledgements

First, i would like to thank my supervisor, Ing. Vít Hlinovský, CSc. and doc. Ing. Jiří Vašíček, CSc. for their consultations and aid, which made creation of this thesis possible.

Many thanks also goes to my friends and colleagues of the eForce FEE Prague Formula team, namely Jan Mánek, Martin Cejp, and Bc. Adam Podhrázský, for their support in development of first working prototypes of the devices, alongside with many design cues for the designs.

Also a thanks goes to my friend, Ing. Jiří Svatoň for his help in understanding of satellite navigation systems and to greater extent entire RF-based technologies coupled with aid in selection of an appropriate sat-nav solution.

And last, but certainly not least, i thank my girlfriend, my closest family and my friends for their unyielding support in my study endeavors.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the *Guideline for adhering to ethical principles when elaborating an academic final thesis*.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work.

Such persons are entitled to use the Work for non-profit purposes only, in any way that does not detract from its value.

This authorization is not limited in terms of time, location and quantity.

.....

Stanislav Tomášek

In Prague, 25. May 2017

Abstract

This thesis describes development of a modular data-logger with an ability to monitor arbitrary systems, data storage and remote transmission.

Thesis is structured into 5 major chapters, each giving overview of a key aspect in the development.

First chapter concerns with development history and introduction to Formula Student competition within the context of development.

Second chapter describes theory of operation alongside concepts of modularity and abstraction used in the device.

Third chapter gives overview of the complete concept development from the schematics to the software.

Fourth chapter explores a potential application of the device in automotive field.

Last chapter is the economical analysis which examine costs of the device final development, based on previous case study, leading to market entry and subsequent production life.

Keywords: modularity, abstrakce, telemetry, data-logging, abstraction, Wi-Fi, GNSS, DPS, CAN bus, STM32, Java, Formula Student, eForce, cost analysis

Supervisor: Ing. Vít Hlinovský, CSc.

Abstrakt

Tato závěrečná práce popisuje vývoj modulárního zařízení pro sběr dat se schopností sledovat libovolné systémy a tyto data jak ukládat, tak i přenášet za pomocí bezdrátových technologií.

Práce je rozdělena do 5 částí, kdy každá dává vhled do klíčových aspektů vývoje.

První kapitola se zabývá historií vývoje a představením soutěže Formula Student v kontextu samotného vývoje zařízení.

Druhá kapitola má za úkol seznámi čtenáře s teoretickým základem funkčnosti zařízení, především koncepty modularity a abstrakce, které jsou podstatnými články fungování zařízení.

Třetí kapitola detailně popisuje navržené zařízení od prvotních schémat až po vývoj softwaru.

Čtvrtá kapitola prozkoumává další potenciální aplikaci zařízení v oblasti *automotive*.

Poslední kapitola se zabývá ekonomickou analýzou nákladů na jak dokončení vývoje zařízení dle případové studie z předchozí kapitoly, tak následnou sériovou výrobu zařízení.

Klíčová slova: modularita, abstrakce, telemetrie, sběr dat, Wi-Fi, GNSS, PCB, CAN, STM32, Java, Formula Student, eForce, analýza nákladů

Překlad názvu: Modulární zařízení pro telemetrii a sběr dat z libovolných systémů

Contents

Figures

Tables

Chapter 1

Introduction

"Data-logger is a device which collates and stores measured data from various sensors and/or other devices for purposes of further processing."

Design objective was to create a complete solution[1] taking it from the initial concept, architecture, schematic and printed circuit design, firmware and software implementation and the economical analysis. The data-loggers were verified with cooperation of the *eForce FEE Prague Formula* team.

First chapter of the thesis will explore the history of the data-logger development in the *eForce* team and the alternative devices we considered or used against/before the proposed design. Chapter will also give overview of the *eForce* team, *Formula Student* competition and vehicles on which the development took its course.

Second chapter will introduce the reader to the theoretical framework and the design of the device along with its architecture. Focus will be on both hardware and software abstraction concepts which allow the device to achieve its modularity.

Third chapter describes actual reference implementation both of the device's hardware and software based on the theory and design from the previous

[1] Designed data-logger will be called in the thesis as "*the device*" (unless specifying other device as subject). In code/schematics/PCBs the device is called *"WiFiLog"* or *"WiFiCAN"*, which is obsolete naming due to not taking its modular design into account.

chapter. Hardware implementation part will range from schematic design to printed circuit board design description. Firmware implementation part concerns with communication of the device with the outside world. Software implementation part will describe developed application for data fetching and study.

Fourth chapter will explore more potential uses for the device by exploiting its modular architecture coupled with the abstraction and openness of the software specification.

Last chapter analyses the economical part of the device. Specifically examination of costs bound with design completion into final, manufacturable form and manufacture itself.

1.1 Project history

The original idea of the telemetry sprung from the placement of a Wi-Fi module at one of the third generation vehicle's *ECU* called *ECUF*. This module was not initially considered for the data-logging because of the *Magneti Marelli's* racing data-logger *HRDL-1* was used instead.

First experiments with the Wi-Fi module and ECUF had shown that we can indeed transfer some of the data on the vehicle's CAN bus onto a remotely connected computer. Unfortunately ECUF's microcontroller didn't possess enough computing power to service the Wi-Fi module properly and caused performance issues during vehicle operation.

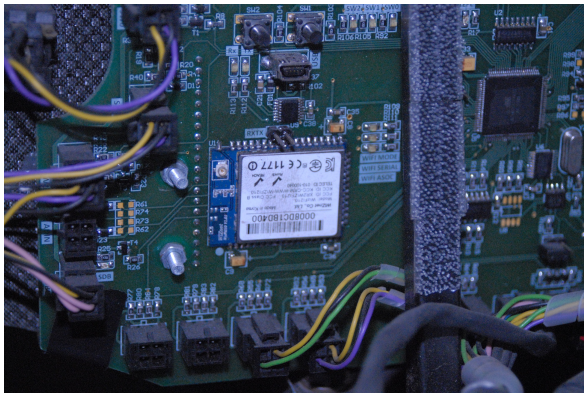


Figure 1.1: Photograph of the Wi-Fi *(WiZFi210)* module present on the *FSE.03*'s ECUF.

During these first experiments we have started on a development of a Java-based application for data processing from data-logger(s). Initially it was only a simple network communication software.

I've further improved it's design and currently it is able to be readily extended with user-made plugins in order to capture more types of data and communicate with various data-loggers.

This application's architecture even started to dictate further direction for the data-logger's hardware. Software will be discussed more in-depth later in this thesis.

1.1.1 WiFiCAN module

The next step was to create a separate unit to handle the telemetry in the vehicle. This device intercepted and captured CAN bus messages, then transformed them into suitable form for further processing.

Device itself was a module for a new revision of the ECUF unit. This meant that the telemetry unit did not needed a separate enclosure or a separate (external) connector, thus reducing it's price and complexity.

During the development a *SD card* slot was added to allow for offline storage of the data in the case of remote connection failure. The *GNSS* and accelerometer chip emplacements were added for precise position information.[2]

It was first fielded on the fourth generation formula car. While telemetry was not available during the actual racing, stored data was equally important in getting much needed insight into the vehicle's behavior during the race.

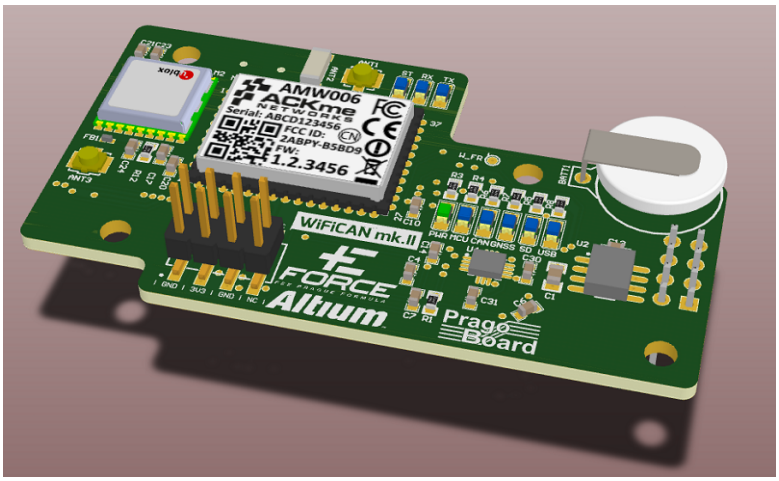


Figure 1.2: Visualization of the *WiFiCAN* data-logger module.

Besides creating a team-made data-logging solution for our vehicle, this unit was also revolutionary by introducing *ARM-based* micro-controllers into future generations of our vehicles' units, in a bid to replace aging *Atmel* micro-controllers.

[2]Not present on the prototype board due to creation of the *ECUG* board.[18]

1.1.2 Custom device

After proving our capability for developing own data-logging/telemetry solution a decision was reached to create dedicated device which will exactly fulfill our needs for data capture, thus eliminating dependence on commercial products. New device needed to fulfill following requirements:

1. Ability to capture data from every team's vehicle with only minor changes to the device.
2. Simultaneous capture of at least 2 CAN buses.
3. Other interfaces for possible sensors on the vehicle
4. Data transfer via Wi-Fi or other wireless technology.
5. Form-factor comparable with *Magneti Marelli's* data-logger.
6. Low production cost

1.2 On FSAE and eForce

This chapter will introduce reader to the *Formula SAE Student* competition and the *eForce FEE Prague Formula* team – the main reasons for the development of this device.

1.2.1 Formula SAE Student competitions

A family of an engineering student competitions held annually in many countries across the globe. The objective is to (every year) develop and build a Formula-class vehicle according to competitions' rules and compete against other student teams under supervision of experts from the automotive, engineering and business fields.

Individual competitions are split into events. These are categorized into two sections – static and dynamic. Static events are concerned with presentation of the vehicle both on engineering and business grounds. Dynamic events are tasked with stressing the vehicle in the actual racing. Each event has a predefined weight in points, team claiming highest sum of said points is the winner. Altogether the official competitions are aggregated by the *WRL* in which every participating team is ranked against other teams.

Competitions' primary objective is to provide engineering students interested in motorsport and technology with a practical experience on the engineering design, hands-on development, project presentation and teamwork.

To this date, more than 600 teams from entire world take part in more than 10 instances of the competition.

History

Original competition was founded in the early 1980s' by the Society of Automotive Engineers in the USA. During the 1990s' competition "arrived" into the continental Europe and many new teams originated, primarily in the Germany and neighboring countries.

Originally only a combustion drivetrain vehicles were eligible for entering. In the 2010 a decision was made for allowing vehicles with electric drivetrains to enter the competition to address a renewed interest of the automotive industry in the electric vehicles.

Rules

In order to specify requirements for vehicles designed and manufactured by the teams and the course of the competitions, a set of a official rules is released every year. By modifying and releasing them every year the competitions stays "fresh" even for the advanced teams and forces them to continue their development, much like the less advanced teams.

Comparing FSAE rules against other (professional) formula-based motorsport rules e.g. the official *Formula F1* rules, the FSAE rules are far less restrictive to the actual design of the vehicles and are focused primarily on securing safety of the competitions' participants.

1.2.2 eForce FEE Prague Formula

History

The first FSAE team at the Czech Technical University in Prague was the *CTU CarTech* under the Faculty of Mechanical Engineering, established in the 2009.

Electric "sister" team, the *CTU CarTech Electric* was established as a reaction to concession of the electric drivetrain in the competition at 2010 under the Faculty of Electrical Engineering, more precisely the Department of Electric Drives and Traction.

Since then the team had split into *CTU CarTech* and *eForce FEE Prague Formula* teams to define boundary between combustion and electric drivetrains more clearly.

Team

The eForce team is consisting of a roughly 30 students from both FME and FEE of both bachelor and master study programmes. Team members are classified into 4 groups according to their specialization and/or interest – Mechanical systems (*MES*), Electrical systems (*ELS*), Project group (*PRG*) and the IT group.

Each group has a designated leader who is answering directly to the *Team captain*. The "buffer" between the team and the faculty is the *Faculty adviser*.

To this date, the team had built 6 vehicles and successfully competed with them across the Europe and the America. The eForce stays the only Czech team building FSAE-class electric drivetrain vehicles.

One of the greatest eForce's strengths is the in-house research and development of every vehicles' part, thus giving the team competitive edge over other teams when presenting the engineering design, adaptability to changing conditions during competitions or troubleshooting. This attitude was one of the impulses, which prompted the team for development of this custom data-logger.

More information about the FSAE competitions, its rules and the eForce team can be found on their official website.[8]

1.3 Solution comparison

Following chapter will introduce reader to products which can be considered as an alternatives or competition for our design.

Every device will be discussed along with their pros and cons and reason for using/not using them in our vehicles.

A quick-overview table for discussed data-loggers can be found at the end of this chapter.

1.3.1 Magneti Marelli's "HRDL-1" series

Data-logger made by Italian high-tech racing parts producer/designer company *Magneti Marelli*.[26]

Unfortunately the data-logger's documentation is considered confidential and cannot be distributed alongside this thesis for further reference.

This data-logger was used by our team in the third generation vehicle. It's compact, lightweight and possesses many features required by motorsport users. Among them – CAN bus logging, lap-trigger and various analog (e.g. temperature sensors) and digital inputs (Ethernet[3], RS-232) for arbitrary signal connection.

Captured data is stored onto local non-volatile memory and retrieved offline after racing event (or testing) end. Manufacturer provides dedicated application for data extraction and analysis.

Primary concern with using this data-logger was that CAN bus capture needs a special configuration done exclusively by the manufacturer (or adoption of manufacturer's protocol). Because majority of our vehicles' workings is governed on the CAN bus and both possibilities of problem resolution were unacceptable for our team. Thus this data-logger was deemed unsuitable for following vehicle generations.

[3] for data download

While not exactly suitable for our needs, many design cues were taken from this data-logger for further development of our device.

1.3.2 Vector Informatik's "GL1010" series

This data-logger is made by German company *Vector Informatik* which specializes in automotive electronics.[11]

This device is targeted primarily on the vehicle manufacturers, therefore not exactly designed for motorsport use. It allows for simultaneous monitoring of 2 CAN buses with additional ability to capture 2 analog and 4 digital signals. It's CAN bus modules are not limited by a specific protocol usage and therefore is suitable for capturing arbitrary CAN bus communication.

This data-logger captures data onto SD card but due to the waterproof design of this specific series it is not removable. Thus the data transfer is limited to the usage of USB cable.

The biggest advantage of this data-logger is a very extensive software support. Our team uses the *CANoe* software supplied directly by the manufacturer.

It is being used by our team as a backup/comparison for our data-logger on the fourth (and later) generation vehicle and is planned for dual working with our solution until our device is fully operational and tested.

More information about this device can be found in manufacturer's official documentation.[10]

1.3.3 Comparison table

Product	*HRDL-1*	*GL1010*	Custom device[4]
CAN Bus interface	1×[5]	2×	2×
Extensible	NO	NO	YES
Data transfer method	Ethernet	USB	WiFi/USB
Offline storage size	Up to 512MB	Up to 32GB SD card	2GB
Other (digital) interfaces	RS-232, ARCNet	None	RS-485, IsoSPI

Table 1.1: Solution comparison table

[4]Reference eForce design
[5]Only specified protocols

Chapter 2

Design

This chapter will introduce reader to the proposed design alongside with the necessary theoretical framework used in the thesis. It also provides an overview of used technologies and the device's overall architecture.

2.1 Technologies

2.1.1 OSI model

In order to understand later discussed technologies and to lesser extent inspiration to the device's architecture, reader should be acquainted with the OSI networking model.

It is also referred to as "ISO/OSI model" in recognition of the *International Standards Organization* which originally drafted the model in 1970's.

Model itself consists of 7 layers. These layers split complex communication systems into more manageable and flexible form.

Communication through ISO model is conceptualized on the figure below. Data enters at specified layer and must go "down" through all layers below

which modify/encapsulate data for further layer until reaching physical layer. After going through physical layer, the data is de-encapsulated by going "up" the model.

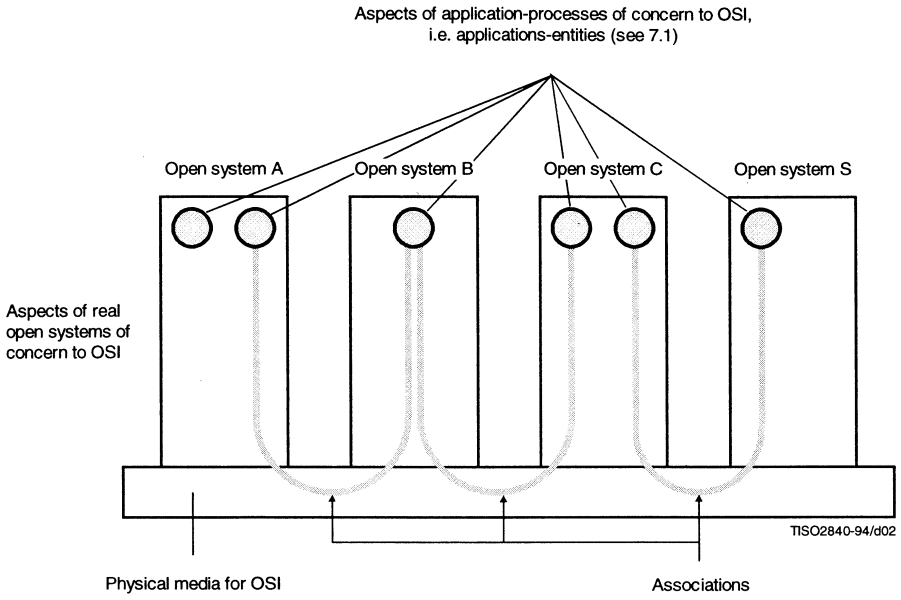


Figure 2.1: OSI model compliant systems communicating through physical medium[14]

Details on individual layers and their functions can be found here.[21] More complex description of the model can be found in the official specification.[14]

The device is using same layered architecture for its modular capabilities.

2.1.2 CAN bus

CAN bus is widely used in automotive and industrial applications for message-oriented, multi-master, fault-tolerant communications. Its primary advantages over other industrial/automotive buses are hardware simplicity and availability, certifications, and available development tools. Maximum standard data rate is 1Mbps, higher data rates can be achieved with newest *CAN FD* standard[15].

It was originally created in 1980s' by Robert Bosch GmbH. Full specification of the bus for both physical layer and link layer can be found freely at Bosch's

website[9].

From the perspective of OSI model introduced beforehand, the CAN bus implements both *Physical* and *Link* layers. Any higher layers must be supplied by the device's implementation.

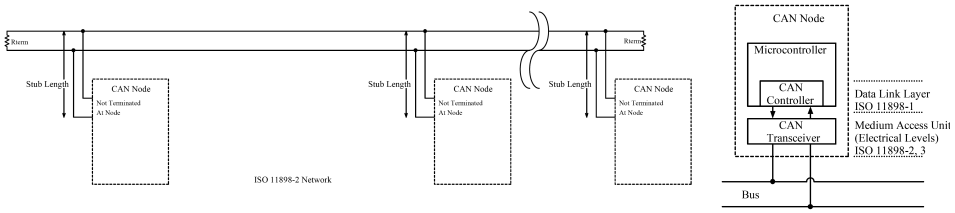


Figure 2.2: Schematic diagrams of high speed network (left) and detail of individual node(s) in the network (right)[33]

The CAN bus transfers data over twisted pair transmission line with $Z_0 = 120\Omega$. Each bus must be terminated on both ends to prevent reflections with $R_{term} = 120\Omega$[1]. Each node require a CAN bus transceiver to transfer from *link layer* to *physical layer*[2] and *vice versa*.

Some microcontrollers contain a CAN bus link-layer interface peripheral, microcontrollers without this peripheral needs another IC to transfer link-layer signals onto different interface. This was one of the team's reasons for switching to usage STM's microcontrollers in their units.

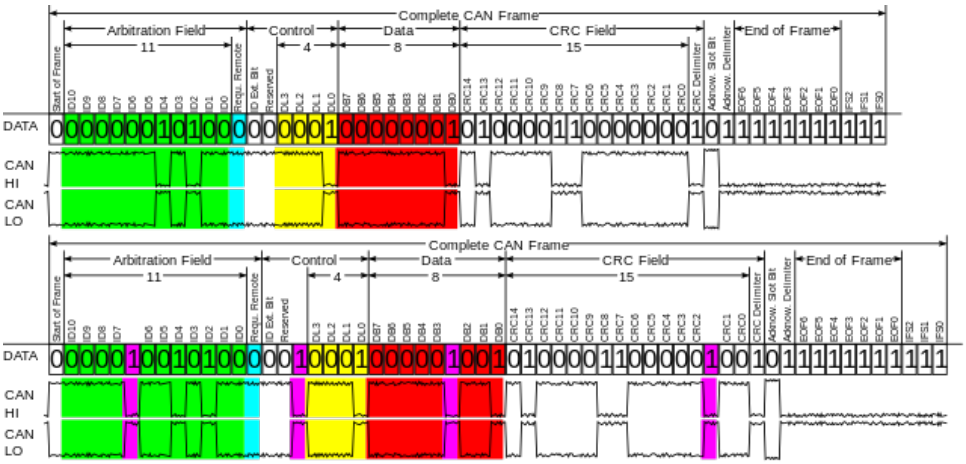


Figure 2.3: Examples of the CAN bus frame[33]. Upper figure lacks stuffing bits (purple)

[1]For high speed CAN bus (ISO 11898-2). Low speed buses (ISO 11898-3) terminate bus at each node.

[2]as per ISO/OSI model notation

Above figure displays a CAN bus frame in its *physical layer* form. The individual bits are represented by two bus states – the *recessive state* (log. 1), which is actively driven by the transceiver and the *dominant state* (log. 0) in which the bus stays dormant.

In order to resolve transmission collisions on the bus a feature called *bus arbitration* is present. An *arbitration field* is present on the beginning[3] of the frame which serves for this purpose. Should two (or more) nodes start transmitting at the same time, they will "fight" for the bus. Each node will successively transfer its message ID and will also listen to the bus state, should the unit transmit recessive bit while bus state is dominant, it will consider its arbitration lost and will stop transmitting until bus is cleared. This necessitates for a unique identifiers for nodes. Simply put, the node transmitting lowest ID value of all nodes will win the arbitration.

Data integrity is verified by every connected node based on the received data and CRC field which is transmitted after the data, to ensure its validity. Should any unit find the data inconsistent with the received CRC, it will drive the bus into a recessive state, signifying invalid message. This feature secures data integrity across the entire bus.

Because the bus does not explicitly transmit a clock signal, a *stuffing bits* of opposite polarity (purple) must be added after successive bits to maintain clock synchronization across the bus.

This information should cover CAN bus basics, further information on the CAN bus can be found here.[33]

The eForce team began to use this bus in their second generation vehicle. Since then, more than 10 units use one bus simultaneously in order to communicate and control the vehicle. Second, auxiliary bus is allocated for dedicated usage by motor controllers & traction control.

2.1.3 Wi-Fi

A wireless local-area network technology based on the specifications from IEEE 802.11 standard and its further amendments.

[3] only for standard CAN frame

The device will utilize Wi-Fi in IEEE 802.11n specification because its availability and favorable characteristics.

It was selected for a wireless data transfer purposes in the device thanks to its "full"[4] ISO/OSI stack implementations availability on the market, reducing time needed and difficulty of development, ability to support sufficient data rates[5] and ability to transfer data over long distances[6].

The ability to reliably transfer data with high relative velocity is discussed in following paper[38]. While paper shows that performance degradation in such situations can be significant for uncompensated device, commercially available devices such as drones can function under these conditions unhampered.

2.2 Architecture

Device's architecture can be split into two intertwining areas – hardware and software. While both use the same idea of *abstraction*, the implementation is quite different in each of those areas.

2.2.1 Packet

To bridge both areas of the device a concept called *packet* was created. It essentially carries data between hardware and software, and *vice versa*. Packets can be thought of as the *quanta* of information in the device.

Packet itself carries information about the source and type of data, *time stamp*, payload length and the payload itself.

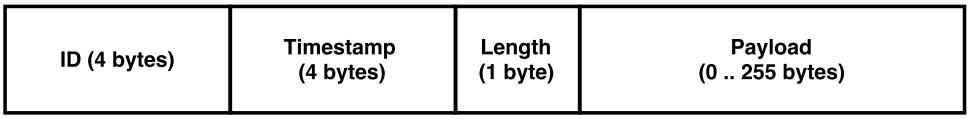


Figure 2.4: Packet structure

[4] Up to the *networking layer*
[5] Up to 600Mbps with selected standard
[6] ~100m with a clear line of sight

- **ID** – Source and type of packet.
- **Timestamp** – Relative[7] date and time of the packet creation.
- **Length** – Size of the following field.
- **Payload** – 0 to 255 bytes of data. Size can be limited by the implementation of the device's firmware.

If the reader is knowledgeable in the area of internet networking, particularly concept of *User Datagram Protocol*, then he can detect a certain similarity.

Individual packets can be further encapsulated individually or *en masse* in other structures which can also provide additional *metadata*.

2.2.2 Hardware modularity

In order to give the potential users ability to use the device in various environments, the device exploits concept known as *modularity*.

The device's hardware is split into three separate PCBs to accommodate this concept. Board exchanging allows for fast change[8] of device's parameters in order to be optimal for intended usage.

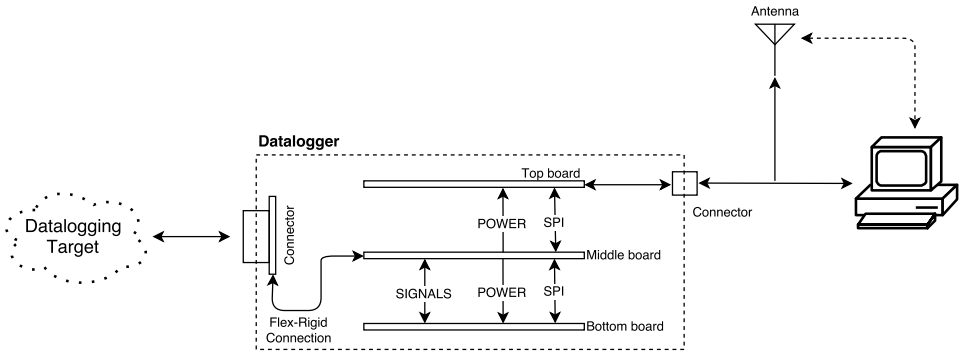


Figure 2.5: Hardware overview of the device

Following sections of the thesis will describe individual boards' roles.

[7] Usually from the start of the device. Origin time is transferred in the encapsulating data container or the special packet.

[8] compared to development and manufacture of a brand new device

Middle board

This board is also called *Base board* in the text. It houses the connector, routes input/output signals to bottom board, converts input voltage into suitable levels for other boards and protects device's circuitry against polarity inversion, over-voltage and over-current. It also acts as a bridge between top and bottom boards.

Bottom board

Also called *Application board*. This board receives signals from the middle board and converts them into packets. These packets are sent via SPI to the Top board. This process can be also reversed – board receives packets from Top board and converts them into signals for the target.

Top board

Also known as *Comm board*. This board houses ICs used for communication with the user's device (e.g. WiFi, Bluetooth, USB) or specialty ICs (e.g. GNSS, accelerometer). This board's responsibility is to process packets both from the Bottom board and (if present) connected user's device. Processed packets can be optionally stored onto local NV memory.

2.2.3 Software modularity

Much like hardware modularity, the software abstraction allows for use of one application for various data-logging targets only by exchanging certain components of the application. Modularity in the application is achieved by abstracting its core workings. These components can be loaded into application by using plugin system.

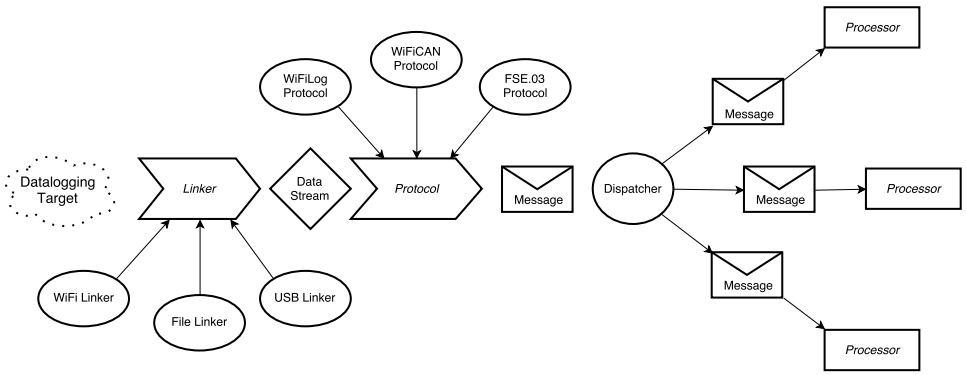


Figure 2.6: Software architecture of the device's application

In this figure, the interfaces' labels are *italic font*, concrete class implementations are represented by normal font.

When describing software's architecture a following story has proven to be quite explanatory:

"Let's consider you came into a foreign speaking country, and you forgot a dictionary. Fortunately you brought your diary and a pencil along. You go to a lecture about this country but while cannot understand a thing, you write down every vowel you hear. Upon returning home you write down everything you heard into special application on your computer and it gives you a translated text."

Same logic of a "translation" can be applied to the application, and in broader scope, the entire device. You plug the data into an application by a keyboard (instance of Linker), data is converted into a stream (feature of Java API[9]) add a correct translation (instance of Protocol) and dispatch the translated data (Messages) to some "Processor" in order to work with them (e.g. draw a graph, save into different format, etc.).

Following section will give more information on the components of the architecture.

[9] Application Interface

Linker

This interface allows for a different types of connection to the target hardware or to process data captured into a file and reviewed. Implementations have to create a link (thus name of the interface) and create the *stream*[10] of data for the next phase.

This interface essentially makes the application invariant to the method of data input. Concrete examples implemented in reference application are shown in the figure 2.6 right below the *Linker* interface step.

Protocol

Concrete implementations of this interface translate from the individual data primitives provided by the stream (bytes, integers, etc.) into the *Messages*.

Implementation must therefore be able to detect message boundaries, resolve endianness, authentication, encryption etc.

Message

Essentially a *Packet* described at 2.2.1. With following differences – timestamp is absolute (done by *Protocol*), payload can be of arbitrary length[11].

Dispatcher

The dispatcher is tasked with taking translated messages from the *Protocol* instance and passing them to registered *Processors*. Each *Processor* has the ability to register (subscribe) itself to receive messages with specified IDs. Dispatcher then constructs lookup table in order to dispatch messages efficiently.

[10] java.io.DataInputStream
[11] limited by the running implementation of Java Virtual Machine

Processor

Concrete processors receive messages from the dispatcher and process them. Processors are bridge between the user and the application. They can also send messages in the opposite direction into the receive-capable devices.

Further information about the software architecture can be found in development manual for the application. [27]

Chapter 3

Implementation

This chapter will give reader overall description of steps undertaken to transform described design into concrete implementation.

Reference implementation is based around CAN bus capture with ability to be readily extended with upcoming *RS-485* enabled sensors in the vehicle. Some analog inputs were established to allow for fast prototyping. Captured data is relayed via Wi-Fi and stored onto device's on-board *NV memory*.

3.1 Hardware implementation

This chapter will introduce reader to PCB design and implementation of hardware blocks in the individual device's boards.

3.1.1 Printed circuit boards

Device's PCBs were designed in *Altium Designer* (versions 15 to 17) software. Altium is widely used by the team in development of vehicles' key electronics due to its relative simplicity over other *EDA* tools, wide usage in the industry[2] and long cooperation between the *Altium* and *eForce*.

Base board

First board to be designed was the base board. This board is specific in its utilization of the *flex-rigid* technology. Primary reason for this design choice is to eliminate the need for wiring inside of the device's enclosure. It also helps the board in other ways – reliability, ease of assembly, aesthetic quality.

Another design challenge is presence of a high current switching power supply on the board. This necessitates for a special care in conductor pad design, proper grounding and component placement for the supply in order to suppress radiated EMI and ensure its valid operation. More details on the power supplies design will be given later in this chapter.

Pitfalls to consider are relative difficulty of both design and manufacture of the board itself. Flex-rigid technology, while being quite old is still comparatively expensive to a "traditional" rigid or flex PCB.

More information on the flex-rigid PCB manufacture can be found in article[13]. While today, EDA tool insufficiency problem is practically nonexistent, technological and manufacturing difficulties persist.

Top board

Next board to be designed was the Top board. Specialty of this board is the usage of 4-layered board for precise impedance control of the PCB traces.

Because of the Wi-Fi and GNSS modules on this board which both utilize high frequency signaling (order of GHz). Impedance of these sensitive traces must be kept within certain bounds in order to ensure proper signals transfer without reflections or losses. Both Wi-Fi and GNSS use lines/traces with $Z_0 = 50\Omega$.

High frequency traces themselves also require special care when routing – no sharp turns (*EMI* problems), ideally no vias (uncontrollable capacitive/reactive impedances), also in order to keep radiated EMI low, vias are scattered all around the border of the board. More resources on the high-speed PCB design which were used in development of this board, can be found at [23][32]

Bottom board

Last board designed was the bottom (user) board. This board has to take care of all data-logger processed signals. It is the most populated PCB of the device because of its high component count required for viable processing of required signals received from the connected vehicle.

3.1.2 Middle board design

As said in previous chapters, the middle board's role is to protect and convert power from the target, route signals to and from the target and other boards.

Top level (block) schematic of the implementation can be seen on the following figure.

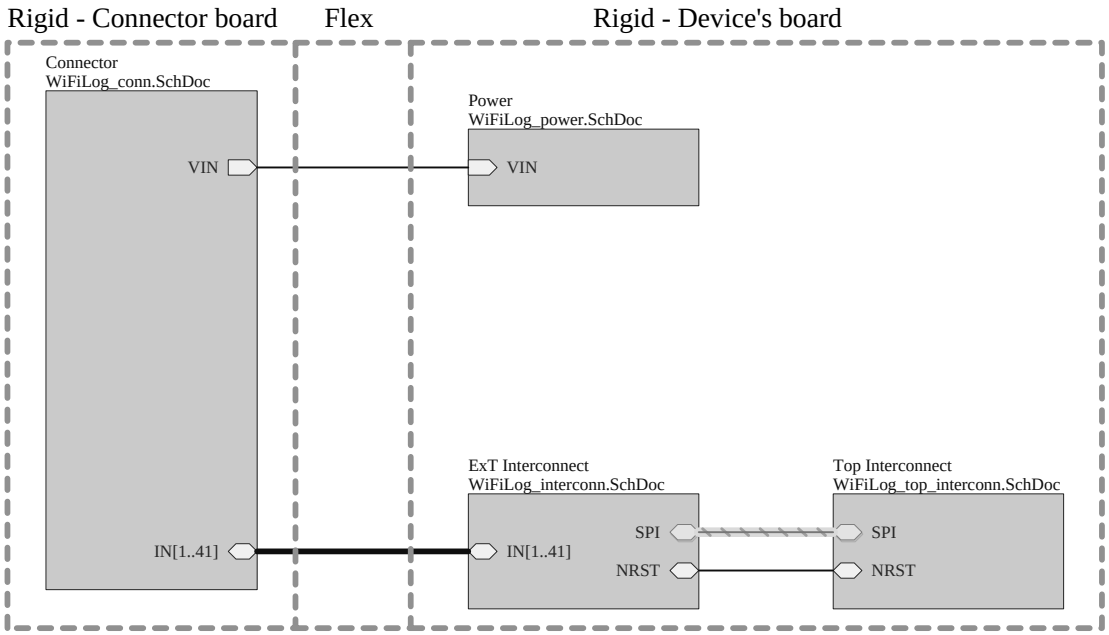


Figure 3.1: Top level schematic of the middle board

Connector block

The *Connector* block contains connector to the target. It is placed on the separate *rigid* board connected to the device's *rigid* board via *flex* section. The block outputs unprotected power and data signals from the connector.

As per eForce team requirements, the connector needs to be qualified for automotive usage. The team used *Souriau* manufactured circular connectors in most of its units[1]. While being relatively expensive connectors, their military standard qualification[36] is a guarantee of reliability, which was lacking in the connectors used by older generation vehicles.

For this implementation, we've selected *Souriau's "8STA-2-12-43"*[36] connector, due to its ability to be directly soldered to the PCB and its pin count, required for connection of needed sensors.

In a commercial manufacture setting, this connector would be replaced by a more available one, such as industry-standard *D-sub* type connector. While not (commonly) offered with a military standard qualification – only with an industrial/automotive standard qualifications, which would be sufficient for most of the device's expected applications.

Power block

This block can be split into 2 parts – protection part and power cascade.

The power protection part is taken from TI's application note[20]. It protects from overvoltage and polarity reversion which are very serious faults with ability to cause serious damage to the device. Undervoltage and overcurrent protections are secured by the power cascade. It also provides soft-start capability to prevent inrush currents.

The power cascade is composed of SMPS for conversion from higher voltages (up to 42V) to 5V and a LDO for subsequent conversion to 3.3V required by digital ICs. Reason for this design choice is that SMPS can operate in high input voltage range and convert them to lower ones with much higher

[1] In present, the team shifts to usage of *Deutsch* connectors.

efficiency than LDOs. While LDOs' voltage conversion range is limited by inherent dissipation-based action of linear regulators' operation.

A *LM22678TJE-5.0*[16] SMPS was selected for its high input voltage swing, high current capability, fixed voltage output without need for external compensation network and automotive qualification for *Q1* component option. Details about the SMPS parameters and a procedure for valid schematic and PCB design can be found in the SMPS's data-sheet.

Primary concern when creating SMPS's PCB design is an emergence of a current loops coupled with high switching frequency. These loops can bring EMI problems when operating sensitive circuitry near them or when conducting EMI testing for various certifications needed by legislation in the case of actual manufacture and introduction of the device onto market.

A *LMS1587IS-3.3*[17] LDO was chosen for its low cost coupled with high current capability and LDO's low EMI noise operation.

Power block also contains 3V battery for RTC backup and operation. This battery would be placed in next revisions directly on the board with RTC microcontroller because removing the board with microcontroller from the base board will cause loss of RTC functionality.

Interconnect blocks

These blocks care about connecting the modular Top board and Bottom board.

Connections between boards are made with standard 2.54mm (100 mil) board-to-board connectors which are typical for stacked PCB interconnection.

Top board connection uses an 8-way, 2-row, SMT, female connector for board power, SPI data connection carrying processed packet communication from connected Bottom board and a device-global reset pin.

Bottom board interconnection uses two 24-way, 2-row, THT, male connector for target signal exchange alongside transformed power, a second connector is a 6-way, 2-row, SMT, male connector which is functionally equivalent to Top board connector (w/o power).

3.1.3 Top board design

Designed implementation of the top board specification is shown on the figure below. This implementation takes the data packets, stores them onto NAND flash memory and sends them via Wi-Fi to a remotely connected equipment. It is also equipped with GNSS chip for precise position information.

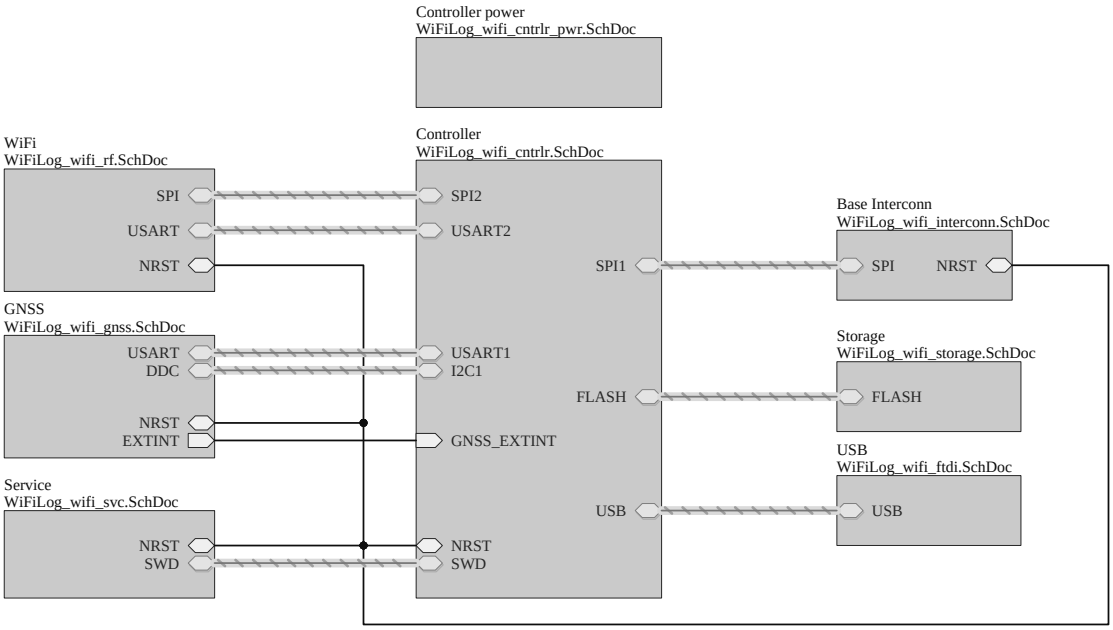


Figure 3.2: Top level schematic of the top board

Controller block

This block contains core of the board – the STM32F205VB[6] microcontroller which takes care of packet processing and routing to peripheral ICs. This microcontroller was selected for its high computing power and peripheral bus availability, most importantly its NAND flash peripheral.

Microcontroller was selected with given criteria using the *STM32CubeMX* software. This software also gives ability to graphically assign pins to peripherals, cutting development time significantly. During software design, it

can also generate setup code. While code generator is only rudimentary and generated code should be reviewed for errors, it can serve as basis for further development.

The microcontroller require external clock sources for precise timing required by NAND flash peripheral, USB and RTC. Two separate clock sources are required – a high speed (*HSE*) for core and peripherals and low speed (*LSE*) for the RTC peripheral. A crystal-based oscillators were selected and designed according to the manufacturer's application note.[3] In newer revision, they will be replaced by electronic oscillators.

Controller power block

Power for the microcontroller requires a 3.3V input power rail, the 1.2V needed by microcontroller's core is created by its internal regulator. This internal regulator requires a two ceramic capacitors for its operation, also a low-ESR, blocking capacitors are recomended for securing smooth power supply voltage in transient conditions on all power inputs. Viable capacitors were selected as per microcontroller's datasheet.[6]

Wi-Fi block

This block houses the *ACKme*'s *AMW006 "Numbat"*[1] Wi-Fi module. This module offers fully implemented TCP/IP stack with a developer-friendly interface.

From hardware standpoint the module itself is comprised of *ARM Cortex-M4* microcontroller and a *BCM43362* Wi-Fi SoC supporting 802.11 b/g/n standards. Implementation only needs to connect power and an antenna for complete function – the module can work on its own without other microcontroller, thanks to its *WiConnect*[4] command interface which can be used to control the module and its peripherals remotely, in the spirit of IoT applications.

Module can communicate with the master controller using a 4-wire UART with maximum declared throughput of 10.5Mbps. Module will[2] also offer

[2] According to module's datasheet, SPI is still not available for data transfer in current firmware implementation

SPI(slave) interface with maximum declared throughput of 21Mbps.

Regardless of interface selection the module's declared data throughput using TCP protocol is 10Mbps which is sufficient for two CAN buses alongside other data from the target vehicle.

GNSS block

The navigation block encapsulates *u-blox's* GNSS IC *MAX-M8W*[29]. This IC was selected to accommodate our requirements on its low-cost, reliability and a competitive parameters.

Due to our lack of previous experiences with GNSS technology and available devices, I've called upon Ing. Jiří Svatoň from the Department of Radio Engineering whose expertise on the GNSS technology was invaluable in selection of a viable solution and giving us clearer insight into this technology. I would like to thank him again for his aid.

This IC provides sufficient position information for data-logging purposes – 4m horizontal precision with 10Hz refreshing frequency for specified worst-case of used GNSS constellations but thanks to the chip's ability to process multiple constellations simultaneously, these characteristics should be considered as absolute borderline.

Chip communicates with the master microcontroller via UART and optionally the *DDC (Display Data Channel)* – a variant of I^2C bus.[28] Module is therefore connected with the primary microcontroller using both UART and I^2C buses. This design choice will allow selection of optimal bus during firmware development – while UART uses GNSS' typical NMEA data format, the *DDC*(I^2C) uses *u-blox's* proprietary format.[30]

Hardware requirements are clearly specified in the module's *Hardware Integration Manual*[28] and were properly followed during actual design. This sensitive hardware was also discussed in section

Storage block

This block encompasses the NV ONFI-compliant NAND flash IC in *TSSOP-48* package[37]. ONFI NAND flash uses synchronous, parallel bus of either 8 or 16 bits in width. Top board hardware implementation uses 8-bit wide bus variant for its reduction in routed pin count.

Usage of a standardized memory interface makes a choice of a particular memory postponable until actual device manufacture and can be exploited to create device variants of differing memory capacities without any need for physical PCB modification.

Hardware implementation must take care to include pull-up and pull-down resistors to control signal pins – the memories' inputs/outputs do not posses these resistors internally and should be left floating, the state machine of the NAND controller can be led into undefined states, rendering the memory unusable.

Service block

Service block contains a 2.54mm programming and reset connector for the microcontroller.

In case of device's mass production, this connector would be omitted and the firmware would be loaded directly to the microcontroller flash memory during manufacture. Firmware updates would be then carried out using different means – most likely through WiFi.

Prototype implementation utilizing this connector, should also contain ESD protection (TVS diode) in order to prevent fatal damage to programming pins.

USB block

This block houses the USB[12] connector for direct interfacing with device development machine.

From hardware standpoint it is a micro-USB AB connector with assorted protection for data lines against ESD damage.

USB is processed directly in the microcontroller by a dedicated peripheral without need for specialized IC. In this instance, the USB will serve for debugging purposes and advanced development of the device.

Because device's enclosure will be sealed during normal operation and shall be opened only during board switching / device servicing, this connector won't be present in the production version of the board.

In this case, the USB's power pin is not used for direct powering the device but for detection of the remote machine connection.

Base interconn block

A counter-connector block for this board's connection to the middle board. See 3.1.2 – "Interconnect blocks" section for further reference.

3.1.4 Bottom board design

As described in theoretical introduction, this board takes care of processing data from the target vehicle. This reference implementation was designed to serve both fourth and fifth generation of the eForce's vehicles. Again, a top level schematic of this board can be seen below.

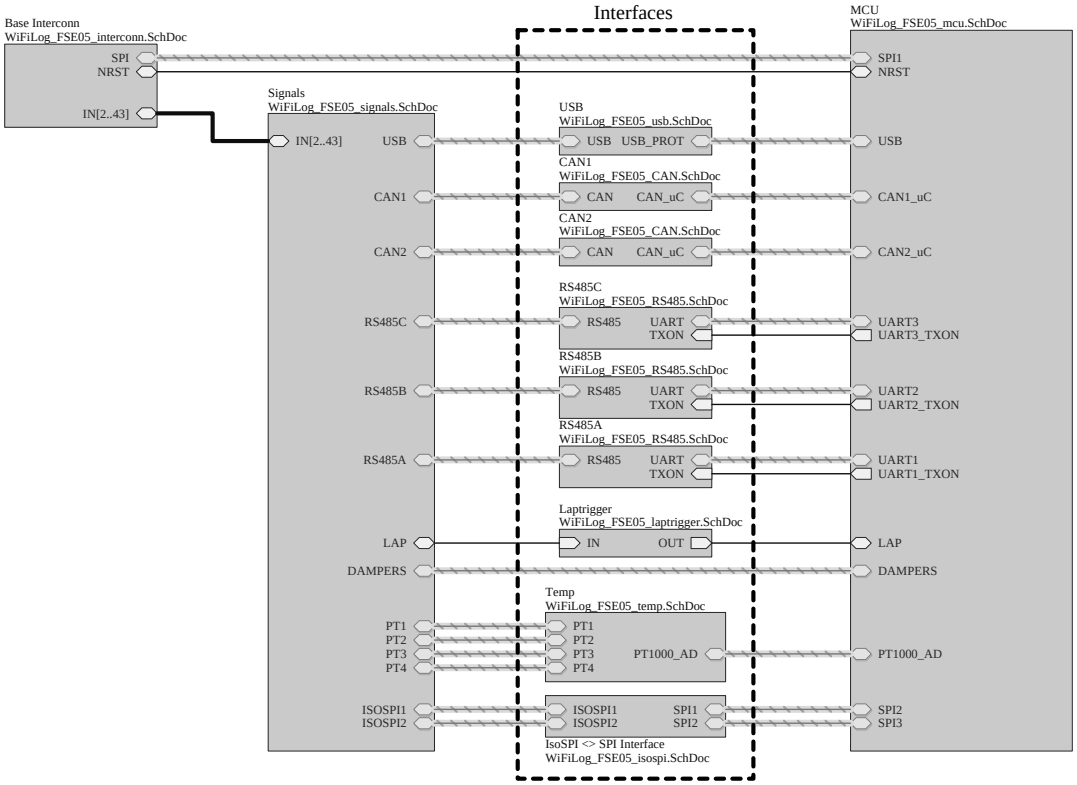


Figure 3.3: Top level schematic of the bottom board

Base interconn block

This block encompasses the board-to-board connectors from the middle board. More details can be found in the 3.1.2 – "Interconnect blocks" section.

Signal block

This block serves as a schematic aid to pick wanted signals received from the middle board. Required signals are taken from the *bus* object and assigned to signal harnesses which are then fed to individual processing blocks.

Interfaces blocks

These blocks take care of various interfaces taken from the connected vehicle. Individual blocks process them into a microcontroller-friendly form.

The primary interfaces are two CAN transceivers which intercept traffic on both buses in the vehicle. Three *RS-485* and two *IsoSPI* transceiver interfaces are prepared for interfacing custom sensors to the device. These interfaces' schematics are done according to the manufacturers' data sheets and application notes.

Lap-trigger block serves as interface for digital input from race lap detector device.

Analog temperature inputs are conditioned by an operating amplifiers in the *Temp* block. I would like to thank Jan Mánek for his design.

3.2 Firmware implementation

The firmware implementation chapter for this device will be strictly theoretical. This chapter will explore possible firmware algorithms and approaches suitable for an effective solution development.

While the newest firmware will be presented only in a theory, firmware of the older data-logger device exists and is being considered as tested and fully functional. It will be used as a reference in order to illustrate workings of a functional data-logger's firmware and since the newest device would draw from the gained know-how.

The *WiFiCAN* board was fitted with similar hardware as the reference device's Top board – the *STM32F2* series microcontroller and the *Ack.me*'s Wi-Fi module.

Flowchart of algorithm used by the *WiFiCAN* device can be found in the enclosed files – see Appendix C.

3.2.1 Initialization

The initialization code for the microcontroller's core is taken directly from the manufacturer's SDK. Code injection is done automatically by the *STM32CubeMX*'s code generator.

Peripheral initialization is done in two layers. The *low-level* initialization takes care of the device's pins. This initialization should be left to the code generator because of sheer number of pins in some packages, it is easy to make mistake in manual assignment and the errors caused can be very tedious to eliminate. On the other side, the *high-level* initialization can be easily done by the developer, thanks to the high quality of the SDK's documentation.

After microcontroller's initialization, other ICs needed to be initialized.

The Wi-Fi module is being initialized by a sequence of *WiConnect* commands. These commands are ASCII text based and are transmitted via UART to the module. Each command is followed by a response from the module, signifying operation's result[3]. The module's configuration can be saved into its NV memory and recalled on a command – this feature can be exploited to significantly reduce setup times.

After initialization, the module is put into the *stream mode* which bypasses the *WiConnect* command interface and allows for seamless transfer of data to connected devices. List of valid commands and configurable parameters can be found in the module's manual.[4]

Other ICs on the *WiFiCAN* board did not need initialization for their basic function.

3.2.2 Run-time

The run-time part of the firmware is composed of two major sections.

[3] If command responses are not disabled

Interrupt

First discussed part will be the (CAN) *interrupts*. Primarily, they take care of the incoming messages from the CAN bus. The captured messages are processed into packets and put into temporary buffer which is being processed by the second firmware part – the *main loop*.

Other interrupts were not established in the prototype implementation. The Wi-Fi module receive interrupt wasn't created due to the security concerns and GNSS and accelerometer ICs wasn't fitted on the prototype.

Main loop

The *main loop* continuously polls for the capacity of the message buffers and in case of being full, it is processed and assigned to be written onto the SD card and sent via Wi-Fi. The interrupt thread's temporary storage buffer is switched to an unused one in order to continue working.

3.2.3 New device

Firmware of the newest device will be split according to the hardware design into two parts for each board with a microcontroller – the Bottom and the Top boards.

Bottom board

As per design, the Bottom boards take care of data processing from the target. Microcontroller's firmware will be primarily consisted of the interrupts which will take care of the data from the target and processing them into packets.

Data transmission to the Top board will be facilitated using DMA peripheral configured to transfer memory contents to the SPI peripheral.

Top board

This board's microcontroller will take on data storage and transmission role. This firmware will be almost identical with the *WiFiCAN*'s implementation, thanks to using same Wi-Fi module and GNSS module supporting NMEA interface.

It will also allow for filtering of transmitted data and ability to receive data from the remotely connected computer in order to tweak the device's (or target's) parameters online.

Again, a DMA will be used for serving SPI peripheral, used by inter-board communication.

3.3 Software implementation

This chapter will present application created for processing of the captured data from the data-loggers.

Due to the device's open specification, the choice of a particular application can be left in the hands of the users. If one chooses not to use this application, it should be relatively easy to adapt a different application (e.g. *Matlab*) for a specific needs.

Custom application was written in the Java programming language. Reason for this design decision was that Java is inherently multiplatform (by using virtual machine-based code execution) thus saving many headaches with forcing seemingly more "powerful" language such as C/C++[19] to behave properly in a multi-platform setting.

Wide platform support is crucial when an objective is to target as many users as possible. Even a team consensus on one platform choice is hard to be found, let alone an agreement in the entire industry.

Java also provides many useful, ready-to-use features such as GUI, networking, dynamical code loading and execution at the developer's fingertips without platform-specific hassle.

The *JavaFX* was selected as GUI library. The *Swing* library is currently being phased out as mainstream Java GUI library.

Weakness to consider is a difficulty with accessing platform-specific or low-level features of executing machine. This can be circumvented by using libraries, which expose needed features. Care must be taken to ensure library's compatibility with host system.

3.3.1 Application description

Following section will describe application based on the actual users' workflow.

Starting-up

The application is started by running `WiFiCAN.JAR` file. Care must be taken to include `lib` directory with application's api JAR file, correct configuration can be taken from the thesis's CD structure – Appendix C.

JAR file can be run by either double-clicking the file in systems with suitable Java binding/configuration or by a following prompt command.

```
java -jar \path\to\app\WiFiCAN.jar
```

The start-up process can take a while[4], use command prompt mentioned beforehand to ensure proper operation from the application log.

After successful start-up, a following window will appear.

Application uses a log to keep accurate track of events during its run. Log can be written into selected file using the *"Dump to file"* button or cleared using the *"Clear App Log"* button.

[4] Up to 20 seconds

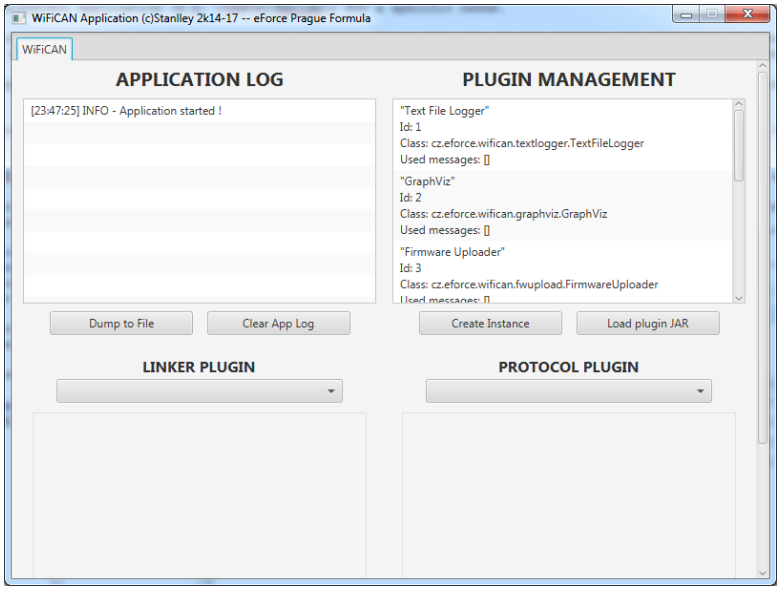


Figure 3.4: Application's main screen

Plugin loading

This feature is used to load plugins containing message definitions or extensions into the running application. After clicking *"Load plugin JAR"*, user can select plugin JAR file(s) to load. If successful, the application's log will note successful loading. In case of failing an appropriate error will be announced and written to the log.

Most common error is loading JAR file without valid plugins for the application, invalid plugin metadata or absence of required message definition(s). Only warning will be generated should user attempts to load already loaded plugins.

Should the user want to load plugins automatically on application start-up a following prompt command can be used to facilitate this. Substitute actual paths to JAR plugin files delimited by semicolon for the `<plugin_files>` placeholder.

```
java -\acrshort{jar} \path\to\app\WiFiCAN.jar <plugin_files>
```

The prompt command can be stored in a batch file for user's convenience.

If plugin JAR contained processor plugin, it will be added to the *"Plugin management"* list. Individual items show processor's name, ID[5], name of the implementing class and a list of used messages. The used message list is used to verify presence of required message definitions.

Connecting to dataloggers

In order to interface application with datalogger, user must select both valid *Linker* and *Protocol* implementations to use for connection creation. Both of these are loaded from plugin JAR files.

After selecting valid *Link* or *Protocol*, they may require more information for connection establishment – e.g. IP address, file path, port. After inputting needed parameters, use *"Connect"* button in order to establish connection.

If the application succeeded in connecting, both Link and Protocol selection will grey out and will be unavailable until disconnected. In case of failure, the user will be prompted by an error dialog to take action.

[5]IDs must be unique in an interface scope – e.g. two implementations of *Protocol* can not use ID 3 but *Protocol* implementation and *Linker* implementation can both be labeled with ID 3

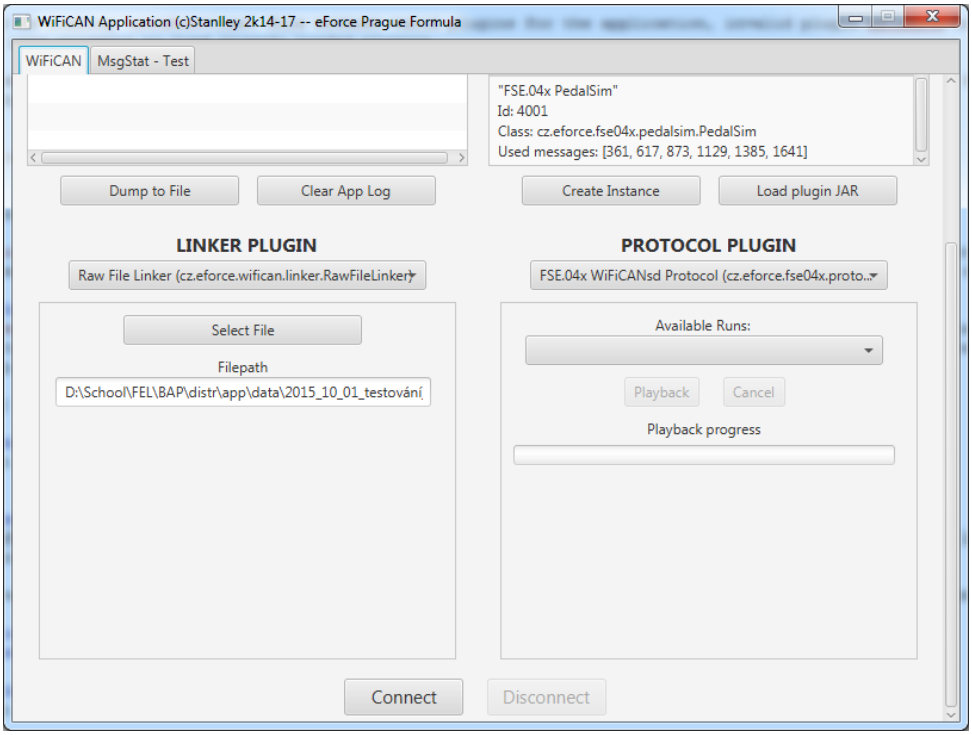


Figure 3.5: Main screen after selecting *Link* and *Protocol*

Processor instantiating

After loading *Processor* plugins, they can be selected from the *"Plugin management"* list and be instantiated using the *"Create Instance"* button. After clicking the button, a prompt will appear with option to enter instance's name which will be shown in the plugin instances tab. Again a success/failure will be reflected in the application's log.

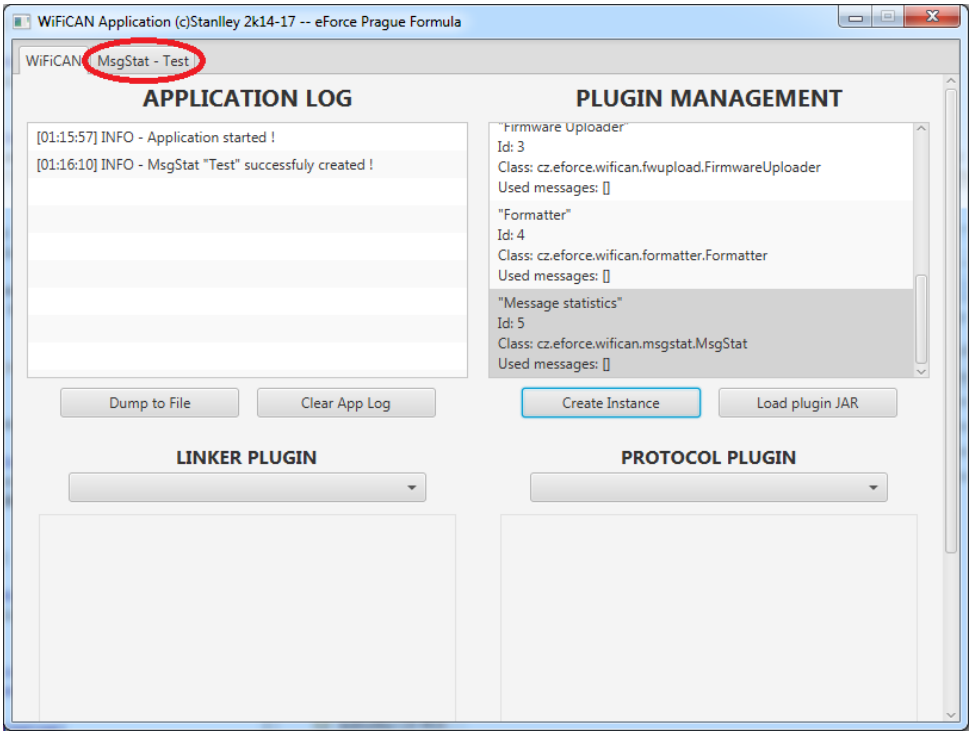


Figure 3.6: Main screen after instantiating plugin named *"Test"*

Processor usage

With processor successfully created, user can select it from the plugin instances tab. Each processor can define its own GUI in order to ensure its functionality and interaction with users.

New plugin development

See 2.2.3 or [27] for further reference.

Chapter 4

Case study

In order to prove described device's usefulness a case study was prepared with an objective to analyze requirements and choosing an optimal approach for fulfilling its needs by using this device.

4.1 Automotive

Beside usage in the eForce's vehicles, the device could be modified in order to interface with current automobiles' OBD interface. This case study would be the most likely candidate for manufacture and marketing of the device.

4.1.1 About OBD

OBD is a family of automotive technologies, devised for vehicles' diagnostics and fault reporting. It is widely used by technicians/vehicle manufacturers/hobbyists in order to troubleshoot vehicles.

While its history going back to late 1960s'. Its implementation is mandatory for all vehicles manufactured and sold in the USA since 1996. EU mandates its implementation since 2003 for both petrol and diesel vehicles present on the market.[34]

4.1.2 Proposed design

Hardware

In order to interface the device with current vehicles, it would require an IC, capable of servicing multitude of automotive-specific interfaces and protocols related to the OBD functionality. Thankfully, the IC with required parameters exists and its specifically designed for said purpose of interfacing microcontrollers with automobiles.

The *STN2120* is a multi-interface, multi-protocol OBD interpreter IC. According to its datasheet, it is able to interface with most of the currently used protocols and interfaces used in automotive industry.

IC communicates with the host processor by *AT commands* via UART interface, making it easy to implement in the solution. Used protocol specification can be found in the chip's manual.[24]

Specific hardware configurations and requirments can be found in the IC's datasheet.[25]

For bridging the selected IC with the Top board, a microcontroller with one UART and one SPI bus can be used. For example a low cost *STM32F030K6* microcontroller.[7]

Firmware

Firmware of the servicing microcontroller would necessitate for implementation of UART coupled with DMA for seamless data transfer from the OBD IC. The interface IC will require certain steps to properly initialize. Vehicle's received data will be processed as specified protocol in the IC's programming manual.[24]

Processed data will be packetized and sent via SPI/DMA combination to the top board for further processing.

Software

Application plugin will have to define all needed message structures. Number of definitions will be

Custom *Protocol* wouldn't need for transmission functionality into the vehicle (not supported by the interface IC).

Processor's functionality will encompass error reporting with an ability to log events into file and contain gauges for fast orientation in values.

Chapter 5

Economical analysis

The economical part of the thesis will analyze the device's manufacturing cost and a market price setting.

Cost analysis will range from the initial setup costs needed for finalizing the design, e.g. optimization, testing, certifications required for sale, to the actual serial manufacture of the device during the active production.

5.1 Initial costs

Initial costs will consist of those needed for finalizing the design. Namely office (and setup), development tools, product optimization, testing and certifications required for sale. These costs were estimated at roughly 2 million CZK over the span of 1 year.

The initial balance sheet is shown in the table below.

Assets		**Liabilities & equity**	
Non-current		Liabilities	
Device development	1 774 857,20 CZK	Initial investment	2 374 857,20 CZK
Computers	100 000,00 CZK		
Software	200 000,00 CZK		
Test hardware	200 000,00 CZK		
Current			
Cash	100 000,00 CZK		
Total	2 374 857,20 CZK		2 374 857,20 CZK

Table 5.1: Initial balance sheet

The device development costs are expanded in the following table.

Wages	
Programmer	740 566,80 CZK
Electro-engineer	714 290,40 CZK
Testing & certifications	
Prototypes	10 000,00 CZK
Testing	30 000,00 CZK
Certifications	30 000,00 CZK
Misc	
Office rent	250 000,00 CZK
Total	1 774 857,20 CZK

Table 5.2: Device development finalization costs

Wages data were taken from the 2016's statistics[22] done by the Czech *Ministry of Labour and Social Affairs*.

Testing and certifications costs were taken from asking people working in the industry.

Office rental costs were researched on the internet through various Real estate agencies.

5.2 Production

The production will take across the span of 5 years, this period is taken from expected lifetime of the product. This will dictate time used for asset depreciation calculations and required cash flows for initial investment amortization.

Cash flow

For this endeavor to be feasible, the sums of yearly cash flows coupled with interest must add up to the initial investment.

The Net Present Value is being used for both ascertaining the project's feasibility and if used in reverse, it can show cash flows (CF_i) needed for investment (NPV) return, based on the interest rate (r).

$$NPV = \sum_{i=0}^{n} \frac{CF_i}{(1+r)^i} \tag{5.1}$$

The interest rate will be is selected from interval of <10% - 20%> for this particular area (15% in analysis document).

Cash flows are considered constant for each year. Final equation is polynomial and its roots are found by using one of the numeric methods available (e.g. *Newton's method*). Fortunately the modern spreadsheet editors contain functionality able to solve such equations.

5.2.1 Product price

Product price will be set using following equation for

$$p = \frac{F}{q} + b \tag{5.2}$$

Where p represents unit price, q is total manufactured quantity (per year), F are (fixed) costs per whole production year and b represents unit cost.

The fixed costs for this scenario are worker wages and asset deprecation.

Unit costs are established from the device schematic documents, by exporting BOM and feeding its constituents to a component database (e.g. *Octopart*). Costs for the PCB manufacture and assembly were taken from offer aggregation website for companies active in this field.

Production quantity must be sufficient in order to (at least) cover for the initial costs spread over production lifetime.

The exact product price calculation alongside calculated value can be found in the analysis document.

5.3 Analysis document

The analysis spreadsheet document can be found at the enclosed CD (Appendix C) with final calculations and graphs relevant for the described project.

Chapter 6

Conclusion

This thesis had presented a development of custom data-logging/telemetry device for the eForce FEE Prague Formula team.

From the initial concept and theory we have explored device's architecture from both hardware and software standpoints and their interconnection.

A concrete hardware, firmware and software design was created for usage in the eForce's formula cars, based on previously described theories and architecture.

An economical analysis explored development and manufacture costs and viability of such activity. Unfortunately, due to lack of data, the comparison against other devices was not done in the thesis.

Described device can be used to collect data from various systems, not only from formula cars, thanks to its modular design in both hardware and software aspects.

Future improvements of the device could target the extensions of available software tools such, refinements of hardware and firmware design and expansion of the device into other fields.

Appendix A

Glossary

C

CAN bus

See 2.1.2. 3–5, 9, 10, 23, 30, 36

D

DMA

Direct Memory Access. A co-processor facilitating memory-memory, memory-peripheral transfers without need for host processor intervention. The host processor specifies parameters (source, destination, data size, burst size, interrupts, ...) and initiates the transfer. Transfer itself is then handled by the DMA controller, leaving the host processor available for another tasks. Host processor is notified about transfer status by an interrupt or by controller's status bit polling. 36, 37, 44

F

FSE

Used in the form of FSE.*ab* – An eForce's vehicle naming system. The FSE part stands for "Formula Student Electric vehicle". Two numbers after the dot specify version of the mentioned vehicle. Newest vehicles equipped with all-wheel-drive are designated with the "x" suffix. For example the *FSE.04x* means "Fourth generation of the Formula Student Electric vehicle with all-wheel-drive". x, 3

G

GNSS

Global Navigation Satellite System. An umbrella term for technologies used for precise geospatial positioning (accuracy within order of meters can be achieved in a civilian receivers) by using timing information from the networks (*constellations*) of Earth-orbiting satellites. Positioning precision and performance can be further augmented by usage of ground-based correction (e.g. *SBAS*, *DGPS*, ...) and/or inertial positioning techniques (accelerometers). 4, 24, 28, 30, 36, 37

L

lap-trigger

Device used for detecting vehicle pass through lap boundary. This is achieved by either optical or magnetic barrier. 9, 34

R

RTC

Real-Time Clock. A device (peripheral) for keeping accurate track of current date and time. Usually backed-up by battery or supercapacitor to ensure operation even during master device's power-down. 27, 29

S

SD card

A standard for a non-volatile storage devices. Specification areas range from cards' physical format to communication interface and protocol. More information on SD cards and standard itself can be found at the official websites of the *SD Association*[5]. 4, 10, 11, 36

SPI

Serial Peripheral Interface bus. A serial, full-duplex, single-master to multi-slave communication bus. Originally specified by the *Motorola* in 1980s. It is being largely used in IC to IC communication[35]. 27, 30, 36, 37, 44

W

Wi-Fi

See 2.1.3. 3, 5, 23, 24, 28, 29, 34–37

Appendix B

Acronyms

B

BOM

Bill Of Materials. 50

C

CRC

Cyclic Redundancy Check. 16

E

ECU

Electronic Control Unit. 3

ECUF

Electronic Control Unit – Front. x, 3, 4

EDA

Electronic Design Automation. 23, 24

EMI

Electro-Magnetic Interference. 24, 27

ESD

Electrostatic Discharge. 31, 32

F

FEE

Faculty of Electrical Engineering (at the Czech Technical University in Prague). 8

FME

Faculty of Mechanical Engineering (at the Czech Technical University in Prague). 8

FSAE

Formula Student SAE. 7

G

GUI

Graphical User Interface. 37, 38, 42

I

I²C

Inter-Integrated Circuit[31]. 30

IC

Integrated Circuit. 15, 26, 28, 30–32, 35, 36, 44, 45, 54

IEEE

Institute of Electrical and Electronics Engineers. 16, 17

IoT

Internet-of-Things. 29

J

JAR

Java Archive file. 38–40

L

LDO

Low-Dropout Regulator. 26, 27

N

NMEA

National Marine Electronics Association. 30, 37

NPV

Net Present Value. 49

NV

Non-Volatile. 23, 31, 35

O

OBD

On-Board Diagnostics. 43, 44

ONFI

Open NAND Flash Interface. 31

OSI

Open Systems Interconnect. 13

P

PCB

Printed Circuit Board. 1, 23–27, 31

S

SAE

Society of Automotive Engineers. 6

SDK

Software Development Kit. 35

SMPS

Switch-Mode Power Supply. 26, 27

SMT

Surface-Mount Technology. 27

SoC

System-on-Chip. 29

T

THT

Through-Hole Technology. 27

TI

Texas Instruments, Inc. 26

U

UART

Universal Asynchronous Receiver and Transmitter. 29, 30, 35, 44

USB

Universal Serial Bus. 31, 32

W

WRL

World Ranking List of the FSAE competition, can be found at `https://mazur-events.de/fs-world/`. 6

Appendix C

CD structure

- `/`
  - `3rd-party/` — third party resources
  - `app/`
    - `bin/` — application in the executable form
      - `lib/`
        - `WiFiCAN-api.jar` — application's public API
      - `WiFiCAN.jar` — core application
      - `WiFiCAN-COM.jar` — COM port interface plugin
      - `WiFiCAN-FSE.03.jar` — *FSE.03* definitions plugin
      - `WiFiCAN-FSE.04.jar` — *FSE.04x* definitions plugin
    - `doc/` — application's documentation
      - `api-javadoc/` — public API *javadoc*
      - `api-reference.pdf` — public API reference manual[27]
    - `src/` — application's source code
    - `test/` — goodies for application demonstration
      - `bin/`
        - `CANServer.jar` — simulator for TCP/IP data transter
      - `data/` — captured data-logger data
      - `src/`
        - `CANServer/` — simulator's source code
  - `doc/` — *this* thesis
    - `src/` — thesis's source code
    - `src/` — thesis's images
  - `eco/` — economical analysis
    - `analysis.ods` — analysis in spreadsheet form
  - `hw/` — hardware design documents
    - `3d/` — 3D-manipulable visualizations

Appendix D

Bibliography

[1] *ADS-MWx06-108R*. AMWx06 'Numbat' Wi-Fi Module Datasheet for use with ZentriOS. ACKme Networks, Inc. (Zentri). 2016.

[2] Syed W. Ali. *Evaluating PCB layout tools: A board developer's perspective*. Nexlogic Technologies. 2011. URL: `http://www.embedded.com/design/prototyping-and-development/4230951/Evaluating-PCB-layout-tools--A-board-developer-s-perspective-` (visited on 12/30/2016).

[3] *AN2867*. Oscillator design guide for STM8S, STM8A and STM32 microcontrollers. STMicroelectronics N.V. 2015.

[4] *ARG-WiConnect-240R*. WiConnect API Reference Guide. ACKme Networks, Inc. (Zentri). 2015.

[5] SD Association. *SD Association Official Websites*. 2017. URL: `https://www.sdcard.org/` (visited on 03/12/2017).

[6] *DS6329*. STM32F205xx – ARM®-based 32-bit MCU datasheet. STMicroelectronics N.V. 2016.

[7] *DS9773*. STM32F0x0 – Value-line ARM®-based 32-bit MCU datasheet. STMicroelectronics N.V. 2017.

[8] eForce FEE Prague Formula. *eForce Official Websites*. 2017. URL: `https://eforce.cvut.cz` (visited on 02/22/2017).

[9] Robert Bosch GmbH. *CAN Specification Version 2.0*. 1991. URL: `http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can2spec.pdf` (visited on 01/10/2017).

[10] Vector Informatik GmbH. *GL Logger Families Product Information*. 2016. URL: `https://vector.com/portal/medien/cmc/info/GL_Logger_ProductInformation_EN.pdf` (visited on 01/10/2017).

[11] Vector Informatik GmbH. *Vector company profile*. URL: https://vector.com/at_company_en.html (visited on 01/10/2017).

[12] USB Implementers Forum, Inc. *Universal Serial Bus Revision 2.0 specification*. 2017. URL: http://www.usb.org/developers/docs/usb20_docs/#usb20spec (visited on 04/29/2017).

[13] John Isaac. "Rigid-flex technology: mainstream use but more complex designs". In: (2007).

[14] *ISO/IEC 7498-1:1994*. 2nd. Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model. ISO/IEC. 1996.

[15] Kent Lennartsson. *Comparing CAN FD with Classical CAN*. 2016. URL: https://www.kvaser.com/wp-content/uploads/2016/10/comparing-can-fd-with-classical-can.pdf.

[16] *LM22678*. 5A SIMPLE SWITCHER, Step-Down Voltage Regulator with Precision Enable. Texas Instruments Inc. 2014.

[17] *LMS1587*. 3A Fixed / Adjustable Output Linear Regulator. Texas Instruments Inc. 2013.

[18] Hostačný Lukáš. *Indirect Speed Measurement for All-wheel-drive Racing Vehicles*. Available from: https://dspace.cvut.cz/handle/10467/64682. 2016. (Visited on 05/15/2017).

[19] Carmine Mangione. *Performance tests show Java as fast as C++*. 1998. URL: http://www.javaworld.com/article/2076593/performance-tests-show-java-as-fast-as-c--.html (visited on 01/08/2017).

[20] Alan Martin. *SNVA717*. Automotive Line Transient Protection Circuit. Texas Instruments Inc. 2014.

[21] Microsoft. *The OSI Model's Seven Layers Defined and Functions Explained*. 2017. URL: https://support.microsoft.com/en-us/help/103884/the-osi-model-s-seven-layers-defined-and-functions-explained (visited on 05/21/2017).

[22] TREXIMA spol. s.r.o.; MSPV ČR. *Informační systém o průměrném výdělku – rok 2016*. 2017.

[23] *SLYP173*. High Speed PCB Layout Techniques. Texas Instruments Inc. 2005.

[24] OBD Solutions. *STN1100 Family Reference and Programming Manual*. 2012. URL: https://www.scantool.net/scantool/downloads/98/stn1100-frpm.pdf (visited on 05/18/2017).

[25] OBD Solutions. *STN2120 – Multiprotocol OBD to UART Interpreter Datasheet*. 2015. URL: https://www.scantool.net/scantool/downloads/206/stn2120-ds.pdf (visited on 05/18/2017).

[26] Magneti Marelli S.p.A. *Magneti Marelli website*. URL: http://www.magnetimarelli.com/company (visited on 01/10/2017).

[27] Stanislav Tomášek. *API Dataloggeru*. eForce FEE Prague Formula. 2016.

[28] *UBX-15030059*. R03. MAX-M8 u-blox M8 concurrent GNSS modules – Hardware Integration Manual. u-blox AG. 2017.

[29] *UBX-15031506*. R02. MAX-M8 u-blox M8 concurrent GNSS modules – Data Sheet. u-blox AG. 2016.

[30] *UBX-16004304*. R04. Addendum to Protocol Specification for HPG 1.30. u-blox AG. 2017.

[31] *UM10204*. Rev.6. I2C-bus specification and user manual. NXP Semiconductors N.V. 2014.

[32] Záhlava Vít. *Návrh a konstrukce desek plošných spojů – Principy a pravidla praktického návrhu*. ISBN: 978-80-7300-266-4. BEN - technická literatura, 2011.

[33] Wikipedia. *CAN bus — Wikipedia, The Free Encyclopedia*. 2017. URL: http://en.wikipedia.org/w/index.php?title=CAN\%20bus&oldid=780001758 (visited on 04/29/2017).

[34] Wikipedia. *On-board diagnostics — Wikipedia, The Free Encyclopedia*. 2017. URL: http://en.wikipedia.org/w/index.php?title=On-board\%20diagnostics&oldid=781254101 (visited on 05/18/2017).

[35] Wikipedia. *Serial Peripheral Interface Bus — Wikipedia, The Free Encyclopedia*. 2017. URL: http://en.wikipedia.org/w/index.php?title=Serial\%20Peripheral\%20Interface\%20Bus&oldid=778275890 (visited on 04/28/2017).

[36] *WMAE8STAPCP03302EN*. 8STA Series PC Tail Contacts. Souriau SAS. 2012.

[37] ONFI Workgroup. *Open NAND Flash Interface Specification*. Intel Corporation, et al. 2014. URL: http://www.onfi.org/~/media/onfi/specs/onfi_4_0-gold.pdf?la=en.

[38] Z. Zhao. "Wi-Fi in high-speed transport communications". In: *2009 9th International Conference on Intelligent Transport Systems Telecommunications, (ITST)*. 2009, pp. 430–434. DOI: 10.1109/ITST.2009.5399314.