SOFTWARE DEVELOPMENT OF A CAN BUS COMMUNICATION INTERFACE USING ESP8266 E MCP 2515

Juliana Carla Santos da Silva[1]; Ricardo Travassos [2].

[1] Graduation Student, Centro Universitário SENAI CIMATEC, BRA.
[2] Professor, Centro Universitário SENAI CIMATEC, BRA.

Abstract: The communication between electrical and electronic modules within a vehicle has been made for some time through the CAN Bus. Due to its bus architecture, it is only necessary a pair of twisted wires in each module for its communication. In this way, all of the modules are integrated, and it saves space and resources by providing savings in the wiring systems. In the market, there is already software to interface this CAN communication, however, none of them supports electronic modules such as MCP2515 (the most accessible hardware option). With that said, the given work's objective is to develop a CAN Bus communication interface using ESP8266 and MCP2515, for the operation of an electric motor.

Keywords: CAN Bus; ESP8266; MCP-2515; EV Powertrain.

DESENVOLVIMENTO DE SOFTWARE DE INTERFACE DE COMUNICAÇÃO EM REDE CAN USANDO ESP-8266 E MCP2515

Resumo: A comunicação entre módulos elétricos e eletrônicos em veículos há muito vêm sendo feita por meio da rede CAN. Devido à sua arquitetura de barramento, são apenas necessários um par de fios trançados em cada módulo para a sua comunicação. Dessa forma todos os módulos ficam integrados, ainda poupando espaço e recursos por proporcionar a economia de fiação. No mercado já existem softwares para essa interface na comunicação CAN, entretanto nenhum suporta módulos eletrônicos como o MCP2515 (opção de hardware mais acessível). Dito isso, o presente trabalho tem como objetivo o desenvolvimento de uma interface de comunicação CAN com ESP8266 e MCP2515, para o acionamento de um motor elétrico.

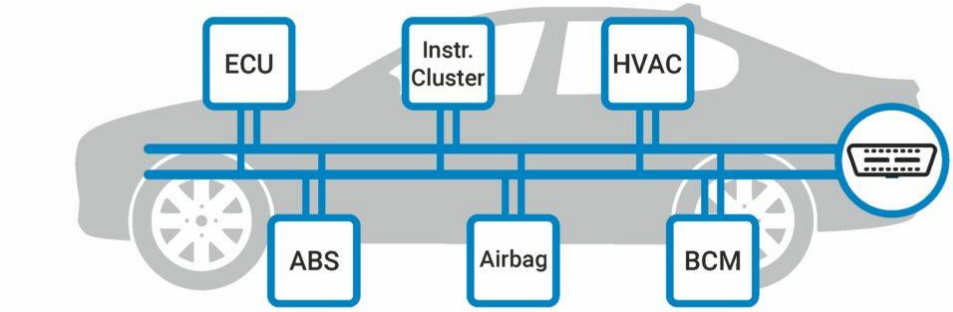
Palavras-chave: Rede CAN; ESP8266; MCP-2515; EV Powertrain.

1. INTRODUCTION

1.1 CAN Network

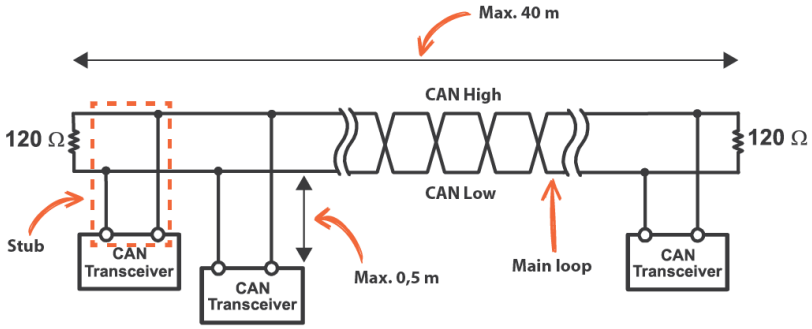
In the past decades, the automotive industry has been growing exponentially, year by year new technology is implemented within the vehicles, and many of the new improvements are based on electronics and software features. To make all this possible, in a car the systems need to be integrated: the sensors, actuators, ECUs, etc. (Figure 1). However, if each of these modules' communication was wired individually, it would overload the electrical whip of the vehicle. To solve this issue, in 1986 Robert Bosch developed the CAN Bus technology, which was later adopted by the ISO11898-1 standard [1,2].

Figure 1. Example of a CAN bus in a vehicle [3].



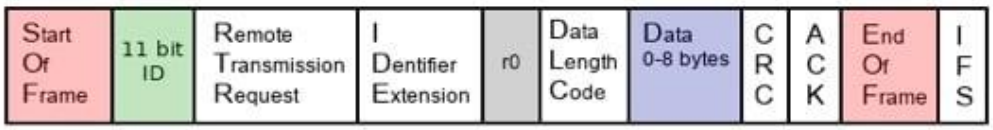
The Controller Area Network is considered a serial communication protocol and it is made on a pair of wires: CAN HIGH and CAN LOW. In this way, the vehicle modules don't need to transmit data individually to its receiver, but instead, transmit it to the network with an exclusively ID for each message which indicates the priority level of the message, and the receiver can filter all the messages in the network by this ID.

Figure 2. CAN bus wiring [1].



The structure of a standard CAN message is distributed as shown in figure 3.

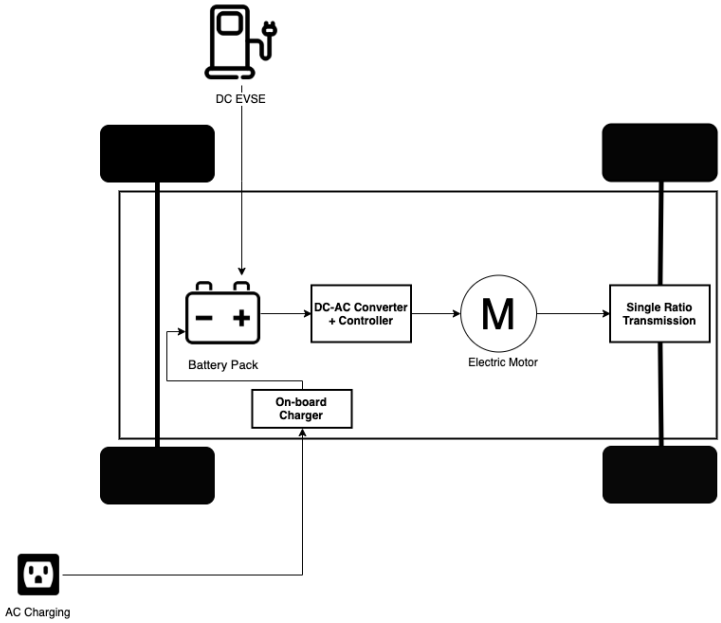
Figure 3. Structure of standard CAN messages [2].



1.2 Electric Powertrain

For environmental concerns, the automotive industry is starting to go through a transformation. Manufacturers are progressively investing in electric vehicles, so it is imperative the study such technologies. For this work, the CAN network will be applied for the activation of an electric DC motor, responsible for the powertrain of a Formula SAE vehicle. This specific model of motor is controlled exclusively by the CAN network, through a series of CAN messages and signals, provided by the manufacturer.

Figure 4. Example of an electric powertrain [4].



1.3 Objectives

Given this, the general objective of this work is to develop a CAN Bus communication interface using MCP2515 and ESP8266. The specific objectives are: read the RX messages transmitted by the BRM ECU; send TX messages to the BRM ECU; and transform the raw data transmitted, into valuable information such as actual speed, actual torque, operation mode, and ignition state.

2. METHODOLOGY

2.1 Boost Recuperation Machine (BRM)

2.1.1 Overview

The electric DC motor used in this experiment is a Boost Recuperation Machine (BRM) from SEG Automotive. Inside the BRM there is a control unit that contains the software used in the motor operations. The BRM needs to receive control signals via CAN to be able to operate properly and also communicate via CAN signals, so it needs to be part of a network. A signal is part of a CAN message and there could be many signals within a message, this distribution is according to each device. In the case of the BRM, the manufacturer provided a list and description of all CAN messages and their signals in the Software Description document. For the purpose of this work, it will be addressed only the main messages for the BRM operation: Pt_Veh_1; Pt_Edrv_Des_1; Eem_Edrv_Lim_1; Edrv_Act_1; Edrv_Req_1; Edrv_Pred_1.

Table 1. Description of the received messages from BRM [5].

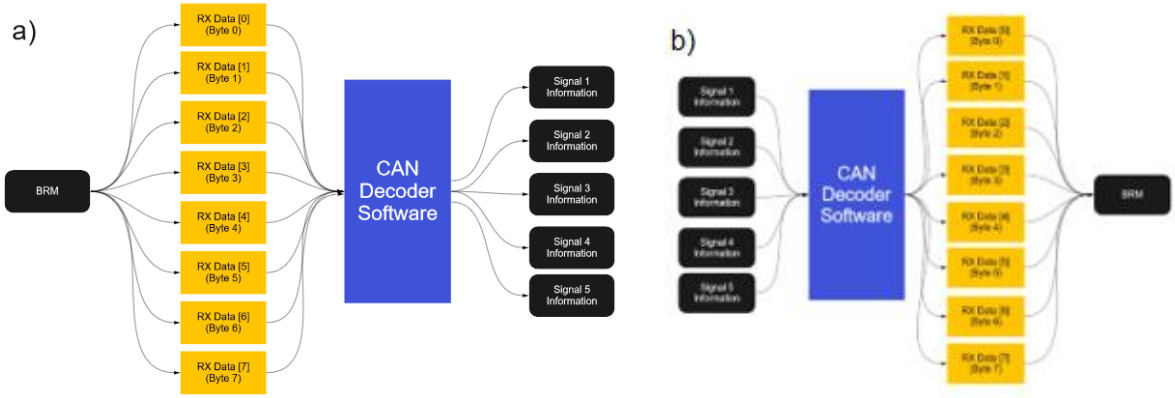
direction BRM view	message	ID / cycle	signal	alias	unit	physical signal range [min/max]		bit witdh	value resolution	offset	default value
TX	Edrv_Act_1	0x2 / 10 ms	Edrv_nrChksAct	*Checksum actual torque*	-	0	255	8	1	0	-
			Edrv_nrAlvCntrAct	*Alive counter actual torque*	-	0	15	4	1	0	-
			Edrv_stRunAct	*Actual run state*	-	0	3	2	1	0	-
			Edrv_stOperModAct	*Actual operation mode*	-	0	15	4	1	0	-
			Edrv_tqAct	*Actual torque*	Nm	-200	209,5	12	0,1	-200	-
			Edrv_nAct	*Actual speed*	rpm	-24000	25146	13	6	-24000	-
			Edrv_uAct	*Actual DC voltage*	V	0	63,9375	10	0,0625	0	-
			Edrv_iAct	*Actual DC current*	A	-500	523	10	1	-500	-
TX	Edrv_Req_1	0x4 / 10 ms	Edrv_nrChksReq	*Checksum request*	-	0	255	8	1	0	-
			Edrv_nrAlvCntrReq	*Alive counter request*	-	0	15	4	1	0	-
			Edrv_tqReq	*Torque calculation EEM*	Nm	-200	209,5	12	0,1	-200	-
			Edrv_nReq	*Speed calculation EEM*	rpm	0	24570	12	6	0	-
			Edrv_stFltId	*Failure identification*	-	0	511	9	1	0	-
			Edrv_flgMilReqd	*Malfunction indicator lamp request*	-	0	1	1	1	0	-
			Edrv_ratTCritMac	*Machine temperature state*	%	0	126	6	2	0	-
			Edrv_ratTCritIvtr	*Inverter temperature state*	%	0	126	6	2	0	-
			Edrv_stTq	*Torque plausibilisation state*	-	0	2	2	1	0	-
			Edrv_stStrtPsbl	*Start possible state*	-	0	3	2	1	0	-
TX	Edrv_Pred_1	0x6 / 10 ms	Edrv_nrChksLim	*Checksum available torque*	-	0	255	8	1	0	-
			Edrv_nrAlvCntrLim	*Alive counter available torque*	-	0	15	4	1	0	-
			Edrv_tqMaxEemPrdn	*Maximum limited torque*	Nm	-200	209,5	12	0,1	-200	-
			Edrv_tqMaxAvl	*Maximum available torque*	Nm	-200	209,5	12	0,1	-200	-
			Edrv_tqMinEemPrdn	*Minimum limited torque*	Nm	-200	209,5	12	0,1	-200	-
			Edrv_tqMinAvl	*Minimum available torque*	Nm	-200	209,5	12	0,1	-200	-
			Edrv_stActDern	*Actual derating state*	-	0	15	4	1	0	-

Table 2. Description of the transmitted messages to BRM [5].

direction BRM view	message	ID / cycle	signal	alias	unit	physical signal range [min/max]		bit width	value resolution	offset	default value
RX	Pt_Edrv_Des_1	0x1 / 10 ms	Pt_nrChksAct	*Checksum torque request*	-	0	255	8	1	0	0
			Pt_nrAlvCntrAct	*Alive counter torque request*	-	0	15	4	1	0	0
			Pt_stOperModDesEdrv	*Operation mode request*	-	0	15	4	1	0	0
			Pt_tqDesEdrv	*Torque request & Maximum allowed torque*	Nm	-200	209,5	12	0,1	-200	0
			Pt_tqMinEdrv	*Minimum allowed torque*	Nm	-200	209,5	12	0,1	-200	-50
			Pt_nDesEdrv	*Speed request*	rpm	-24000	25146	13	6	-24000	0
RX	Eem_Edrv_Lim_1	0xD / 10 ms	Eem_uMaxEdrv	*Maximum allowed DC voltage*	V	0	63,9375	10	0,0625	0	52
			Eem_uMinEdrv	*Minimum allowed DC voltage*	V	0	63,9375	10	0,0625	0	36
			Eem_rMvbEdrv	*Estimated 48V-battery resistance*	Ohm	0	1,023	10	0,001	0	0,02
			Eem_iReqEdrv	*Current calculation EEM*	A	-500	523	10	1	-500	0
			Eem_iMaxEdrv	*Maximum allowed DC current*	A	0	1023	10	1	0	250
			Eem_iMinEdrv	*Minimum allowed DC current*	A	-1000	23	10	1	-1000	-250
RX	Pt_Veh_1	0x10 / 100 ms	Pt_nrChksIgn	*Checksum ignition switch state*	-	0	255	8	1	0	0
			Pt_nrAlvCntrIgn	*Alive counter ignition switch state*	-	0	15	4	1	0	0
			Pt_stIgnSwt	*Ignition switch state*	-	0	7	3	1	0	4
			Pt_nEng	*Engine rotation speed*	rpm	0	16383	14	1	0	0
			Pt_vVeh	*Displayed vehicle speed*	km/h	0	409	12	0,1	0	0
			Pt_tAmbEdrv	*Ambient temperature E-Drive*	°C	-40	215	8	1	-40	120
			Pt_tiEdrvShOff	*E-Drive shut-off time*	min	0	4095	12	1	0	0
			Pt_flgCrash	*Crash flag*	-	0	1	1	1	0	0

Based on the description in the tables above, it is possible to transform the raw data received by CAN into valuable information and the intended transmitted signals into the raw data to send in 8 bytes. In other words, for reading, the system distributes the 8 bytes received for each message into the signals within it. For transmitting, the system transform the data of each signal into the 8 bytes of the message. Both the reading and transmitting processes happen in a 10-millisecond loop.

Figure 5. BRM communication flowchart. a) Reading process; b) Transmitting process.



2.1.2 Bit distribution

To make the decoding process possible, there are columns in Tables 1 and 2 that make it happen. According to the BRM manual, it uses the Intel byte order (Figure 5). Using the values of bit width, value resolution, and offset of each signal, based on the Intel byte order the software can distribute the signals within the 8 bytes (Figure 6).

Figure 6. Intel Byte Order. [6]

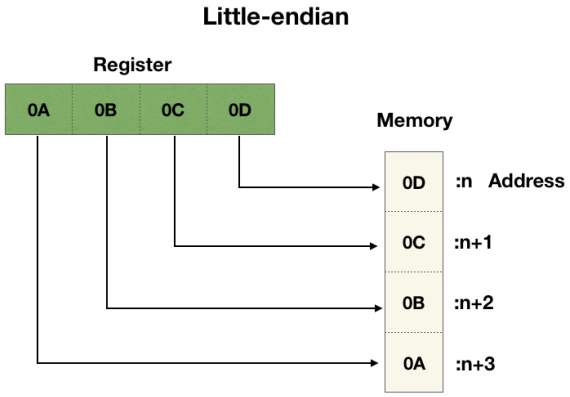
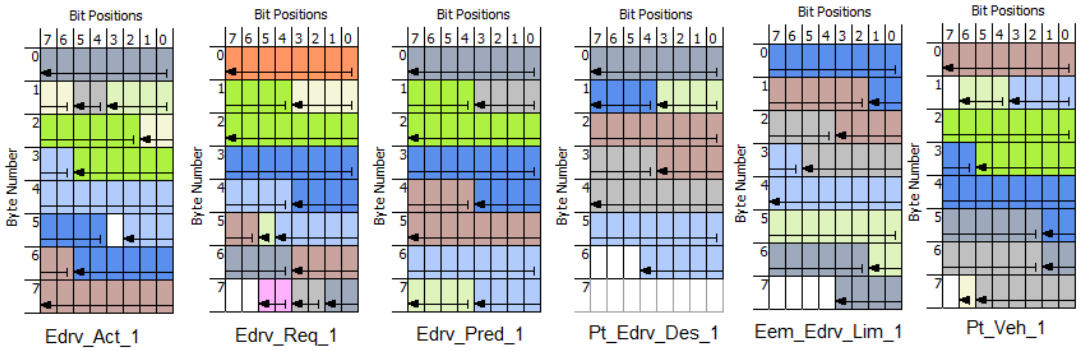


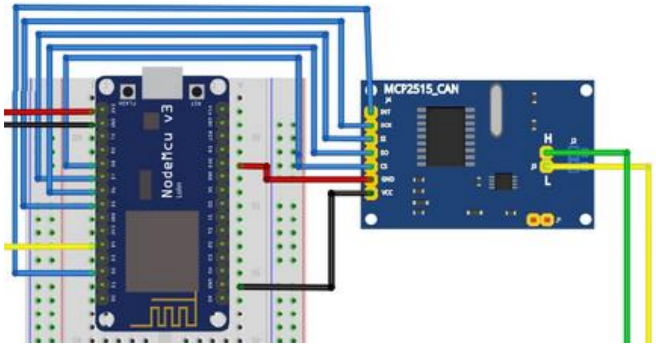
Figure 7. Bit distribution of the main messages addressed in this work.



2.2 ESP-8266 and MCP2515

The hardware used in this work to communicate with the BRM ECU is a combination of the NodeMCU ESP8266 Board and the MCP2515 CAN Module (Figure 8), due to the low costs of implementation and its ease to use.

Figure 8. Schematics of the circuit [7].



2.3 Hardware embedded coding

The hardware code was developed in the Arduino IDE, using the libraries: <stdio.h>, <stdlib.h>, <math.h>, <SPI.h>, <mcp2515.h>. After the declaration of the libraries, first, the declaration of each message is made, based on the mcp2515 library (Figure 9a). Then it is created a struct ("can_signal") to represent the messages, with all of the valuable information to be used (Figure 9b), and finally created a variable can_signal for each signal within the messages for bit management and distribution (Figure 8).

Figure 9. a) Declaration of CAN messages. **b)** CAN signals struct.

a)

```
struct can_frame Pt_Edrv_Des_1;
struct can_frame Pt_Veh_1;
struct can_frame Edrv_Act_1;
struct can_frame Edrv_Req_1;
struct can_frame Eem_Edrv_Lim_1;
struct can_frame Edrv_Pred_1;
```

b)

```
struct can_signal{
    int data;
    int bit_lenght;
    int data_bin[100];
    int id;
    float resolution;
    int offset;
    String description;
};
```

Figure 10. Declaration of signals within Pt_Veh_1 message according to Fig. 7b struct.

```
//Pt_Veh_1 setup
//data/bit_lenght/data_bin/id/resolution/offset/description
can_signal Pt_nrChksIgn =      {0, 8,  {0}, 0x10,1   ,0,   "Checksum Ignition Switch State"};
can_signal Pt_nrAlvCntrIgn =   {0, 4,  {0}, 0x10,1   ,0,   "Alive Counter Ignition Switch State"};
can_signal Pt_stIgnSwt =       {0, 4,  {0}, 0x10,1   ,0,   "Ignition Switch State"};
can_signal Pt_nEng =           {0, 14, {0}, 0x10,1   ,0,   "Engine Rotation Speed"};
can_signal Pt_vVeh =           {0, 12, {0}, 0x10,0.1,0,   "Displayed Vehicle speed"};
can_signal Pt_tAmbEdrv =       {0, 8,  {0}, 0x10,1   ,-40, "Ambient Temperature E-Drive"};
can_signal Pt_tiEdrvShOff =    {0, 12, {0}, 0x10,1   ,0,   "E-Drive shutoff time"};
can_signal Pt_flgCrash =       {0, 1,  {0}, 0x10,1   ,0,   "Crash Flag"};
```

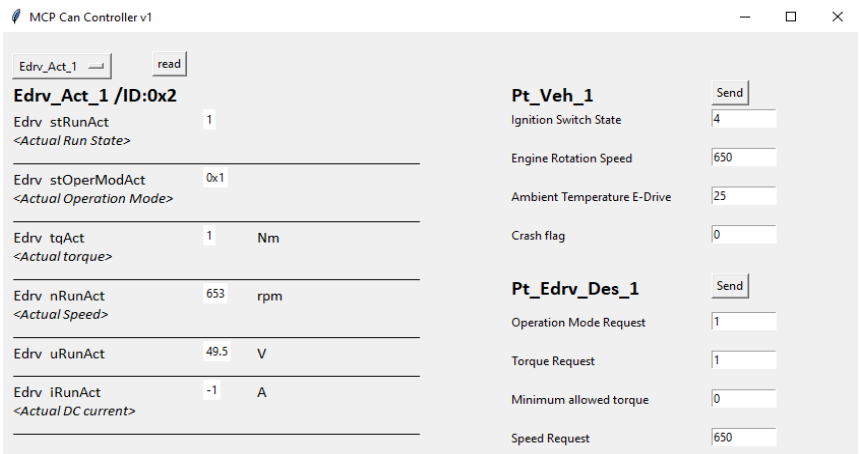
2.4 Graphical User Interface (GUI)

The graphical user software interface was developed in python, based on Tkinter, and PySerial libraries. Tkinter was used to display the interface and PySerial to communicate with the ESP8266 via serial ports. For reading mode, MCP2515 receives the bytes of the message, then, the ESP8266 distributes it into the signals and sends it to python GUI via serial, to be displayed. For transmitting mode, the GUI receives the input of the values to update, sends it to ESP8266, then converts the signals into the 8 bytes message and sends it to the BRM through MCP2515.

3. RESULTS AND DISCUSSION

The first version of the interface is displayed in figure 9. The reading mode is shown on the left side, where you can choose the message to be read through a dropdown menu. While the transmitting mode is shown on the right side of the window, with an entry box for updating each value within the messages. Both processes can be done synchronously.

Figure 11. System Graphic User Interface.



The backend of the application presented efficient results in the tests and was able to activate the motor controlling its speed and ignition state, when applied in Arduino's IDE. The challenge now is to enhance the system's frontend and link it to the backend in order to deliver a user-friendly system, with a proper HMI.

4. CONCLUSION

Nowadays there is already hardware for CAN network communication in the market, but most of them are inaccessible due to their high price. None of the software available for this purpose supports the MCP2515 hardware. The software and hardware integration presented in this work met its objectives and is the begging of a more accessible way to communicate messages in a CAN Bus. For a continuation of this project, it is planned to add more messages to send and read in the GUI and to make it open source once it is finished.

Acknowledgments

To Tec-H2-Racing Hybrid Formula SAE Team, SEG Automotive & SENAI CIMATEC.

5. REFERENCES

[1] ALFERINK, Timon. **Practical tips: CAN-Bus**. KMP Drive Train Solutions, 2017. Available in: < https://www.kmpdrivetrain.com/paddleshift/practical-tips-can-bus/> Acessed: 04 july 2022.

[2] MICHAEL, Stephen. **Introduction to CAN (Controller Area Network)**. All About Circuits, 2019. Available in: <https://www.allaboutcircuits.com/technical-articles/introduction-to-can-controller-area-network/> Acessed: 04 july 2022.

[3] **CAN BUS ANALYSIS**. DevCom. Available in: < https://www.devcom.cz/en/automotive-systems/can-bus-analysis/> Acessed: 04 july 2022.

[4] **Top ev powertrain components manufacturers in india.** EVreporter. Available in: < https://evreporter.com/ev-powertrain-components-manufacturers-in-india/> Access: 4 july 22.

[5] SEG Automotive. **Software description for Boost Recuperation Machine.** 2018.

[6] OTSUKA, Kohei. **Little endian vs Big endian.** 2020. Available in: < https://levelup.gitconnected.com/little-endian-vs-big-endian-eb2a2c3a9135>Access:4july 22.

[7]AL-AREQUI, A., SZAKÁCS, T. **CAN Bus communication demonstration tool for education.** 2021. IEEE International Symposium on Applied Computational Intelligence and Informatics.