ENCODING AND DECODING CONTROLLER AREA NETWORK FRAMES WITH THE USE OF THE CAN DATABASE

Oleg Ivaniuk, PhD, As.-Prof., Halyna Vlakh-Vyhrynovska, PhD, As.-Prof., Roman Modla, PhD, As.-Prof., Nazar Kulyk, student;
Lviv Polytechnic National University, Ukraine,
e-mail: olegiv.mail@gmail.com

https://doi.org/

Abstract. The article examines the features of building a Controller Area Network (CAN) in the automotive industry. The main steps for encoding and decoding physical values in CAN and CAN FD (CAN with flexible data rate) frames are provided. The syntax of messages and signals in CAN DBC has been analyzed. An example of a DBC file that can be used to encode and decode the speed and engine speed of a truck is reviewed. Based on the Linux operating system and the python programming language, an experimental scheme of a virtual controller area network was created, which encodes data on one node and decodes data on the other using CAN DBC.

Key words: Controller area network, Frames encoding, Frames decoding, Virtual network

1. Introduction

Modern control systems and onboard diagnostics of cars are complex systems with a large number of sensors, executive mechanisms, and control units that ensure traffic safety by analyzing the received data and making the necessary decisions [1].

Today, in this industry, various communication technologies are utilized to build a local network between the components of a modern car. A vivid example can be the controller area network protocol, which provides high reliability and an almost unlimited number of nodes in the network [2]. At the same time, despite this protocol's technical characteristics, many higher-level protocols make the standardization of this industry more complicated. Practical studies show that when developing CAN networks (not only in the automotive industry), problems occur with debugging and testing the nodes [3].

At the same time, it is impossible not to note the development trends of the CAN protocol, a new standard (third generation) of the CAN protocol, namely CAN XL, has already been announced. Its application will further escalate the problem of integration and testing of new and existing systems [4].

However, CAN XL should further expand the range of CAN applications and ensure the use of this protocol in systems with a large amount of data [5].

2. Drawbacks

During transport development, many systems for remote diagnostics and control of vehicles operating the CAN protocol have appeared. At the same time, these systems mainly offer complex solutions that often contain unnecessary functionality for the consumer or work at higher levels and do not allow the client or developer to work at the controller area network protocol level.

3. Goal

This article aims to develop an experimental virtual controller area network that enables debugging and testing of the entire system and its nodes.

4. CAN bus specification

If we consider the car as a human body, then the controller area network bus is a nervous system that provides communication. In term of the "nodes" or "electronic control units" (further – ECUs) can be thought of as parts of the body connected via the CAN bus.

So, what is an ECU? In the automotive controller area network bus system, the ECU can be, for example, the engine control unit, airbags, audio system, etc. A modern car can have up to 70 ECUs – and each of them can have information that needs to be shared with other parts of the network.

The controller area network bus allows each ECU to communicate with all other ECUs without complex special wiring. In particular, the ECU can prepare and transmit information (such as sensor data) via the controller area network bus (consisting of two wires, CAN low and CAN high). All other ECUs receive the transmitted data on the controller area network – and each ECU can then examine the data and decide whether to receive or ignore it.

Technically speaking, the controller network is described by the channel layer and the physical layer. In the case of high-speed CAN, ISO 11898-1 describes the channel layer, while ISO 11898-2 represents the physical layer [6, 7, 8]. The role of the controller area network is often represented in the 7-layer OSI model.

The physical layer of the CAN bus defines data such as cable types, electrical signal levels, node requirements, cable impedance, etc. For example, ISO 11898-2 dictates some parameters, including those presented below.

Data transfer rate: controller area network nodes must be connected via a two-wire bus with a transfer rate of up to 1 Mbit/s (classic CAN) or 5 Mbit/s (CAN FD). Cable length: The maximum length of this network cable should be between 500 meters (125 kbps) and 40 meters (1 Mbps Termination: The CAN bus must be properly terminated with a 120-ohm CAN bus termination resistor at each end of the bus.

In the context of networks for automotive automation, we often come across different types of networks. We provide a brief description below.

High-speed CAN bus. The controller area network bus works with the standard mentioned above for the physical layer. This standard provides a data transfer rate in the range from 40 kbit/s to 1 Mbit/s (classical CAN). The standard has become widespread in modern automotive applications, as it enables the construction of a simple cable network. It also became the foundation for several higher-level protocols, including CANopen, OBD2, NMEA 2000, J1939, etc. The second generation of CAN was named CAN FD [9].

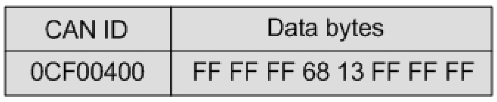
Low-speed CAN bus. The standard is characterized by a data transfer rate in the range from 40 kbit/s to 125 kbit/s and ensures the operation of the CAN bus, even in the event of a fault on one of the two wires. In this system, each CAN node has its CAN termination.

LIN tire. It is a single-wire serial bus that is a cheaper addition to controller area networks. The cheapness of LIN is explained by the fact that the implementation of the LIN protocol is completely software and is built based on the usual asynchronous UART interface. The LIN bus is often used for non-main components of a car, such as an air conditioner, and headlight corrector, as well as for collecting data from simple sensors (temperature, light) [10].

Automotive Ethernet network [11]. It is heavily implemented in the automotive segment to support the high throughput requirements of ADAS driver assistance systems, infotainment systems, cameras, etc. Automotive Ethernet provides significantly higher data rates compared to the CAN bus, but lacks some of the security and performance features of Classical CAN and CAN FD. Automotive Ethernet, CAN FD and CAN XL are predicted to be applied in new automotive and industrial developments shortly.

5. Basic aspects of encoding and decoding of "physical values" in CAN frame

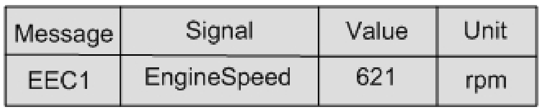
The controller area network database file – CAN DBC, has a text format and contains the data necessary for decoding the raw information of the CAN bus to "physical values" [12]. Consider the term "raw CAN data" based on the example of a CAN frame from a truck (Fig. 1):



CAN ID	Data bytes
0CF00400	FF FF FF 68 13 FF FF FF

Fig. 1. Raw CAN data

Depending on the CAN standard, a frame can contain different amounts of data in two main fields, namely the message identifier and the payload message. In the example (Fig. 1), we are using the standard CAN protocol, since the data length is 8 bytes. If we have a CAN DBC that contains the decoding rules for the CAN ID, we can "extract" the parameters (signals) from the data bytes. One of these signals can be EngineSpeed (Fig. 2):



Message	Signal	Value	Unit
EEC1	EngineSpeed	621	rpm

Fig. 2. Decoded physical values

For a better understanding of the DBC decoding process, let's analyze the DBC syntax and step-by-step decoding examples.

DBC message and signal syntax. First, consider a real example of a controller area network DBC file (Fig. 3).

Above is a J1939 DBC demo file that describes the rules for decoding speed (km/h) and engine speed (rpm). We can create a new text file from the contents of the source file, rename it, for example, j1939.dbc, and utilize it to determine the speed and revolutions of the engine of trucks, tractors, or other heavy-duty vehicles [13]. The DBC file contains the rules (Fig. 4) for decoding the controller area network message and signal (Fig. 4).

Explanation of DBC message syntax [14]:

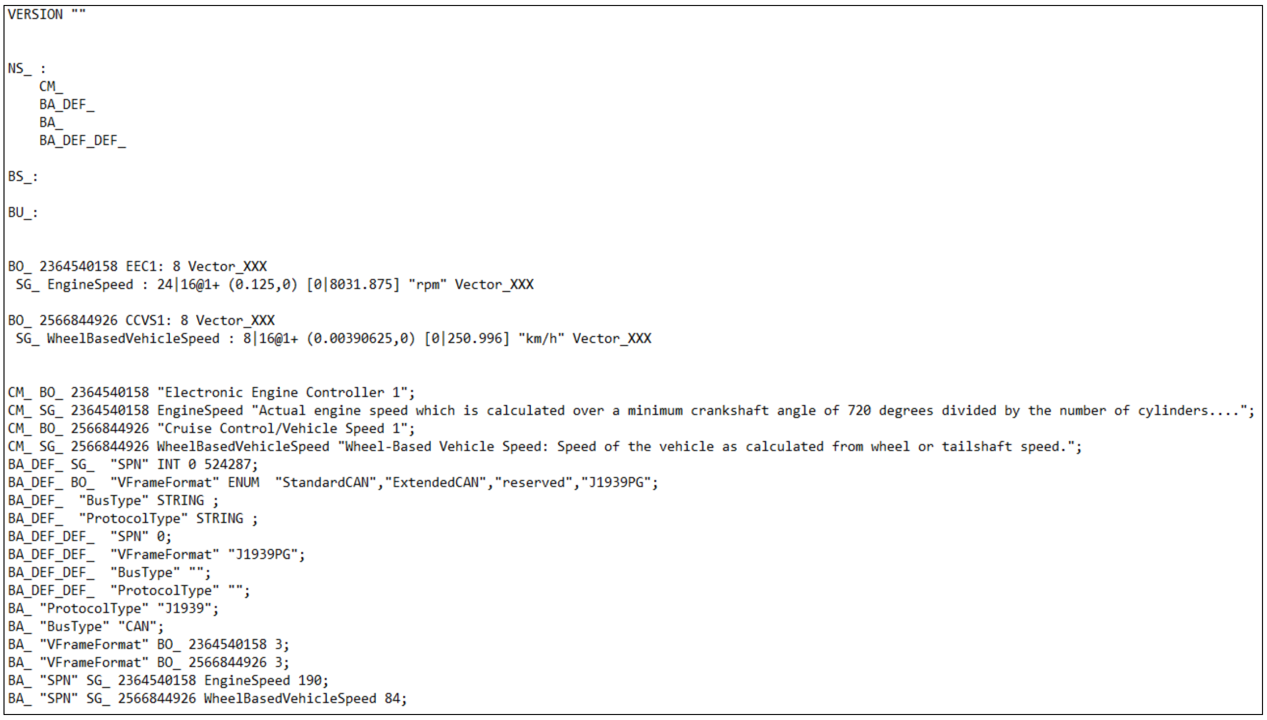
– BO_ is indicated at the beginning of the message, and the identifier must be unique and in decimal (not hexadecimal) representation;

– the DBC identifier adds 3 extra bits for 29-bit CAN identifiers, which are reserved for the "extended identifier" flag;

– the name must be unique, the number from 1 to 32 characters and may also contain letters of the Latin alphabet [A-z], numbers, and an underscore;

– length (DLC) must be an integer from 0 to 1785;

– the name of the transmission node is the sender, or if the name is not available, Vector__XXX is indicated.



```
VERSION ""


NS_ :
    CM_
    BA_DEF_
    BA_
    BA_DEF_DEF_

BS_:

BU_:


BO_ 2364540158 EEC1: 8 Vector_XXX
 SG_ EngineSpeed : 24|16@1+ (0.125,0) [0|8031.875] "rpm" Vector_XXX

BO_ 2566844926 CCVS1: 8 Vector_XXX
 SG_ WheelBasedVehicleSpeed : 8|16@1+ (0.00390625,0) [0|250.996] "km/h" Vector_XXX


CM_ BO_ 2364540158 "Electronic Engine Controller 1";
CM_ SG_ 2364540158 EngineSpeed "Actual engine speed which is calculated over a minimum crankshaft angle of 720 degrees divided by the number of cylinders....";
CM_ BO_ 2566844926 "Cruise Control/Vehicle Speed 1";
CM_ SG_ 2566844926 WheelBasedVehicleSpeed "Wheel-Based Vehicle Speed: Speed of the vehicle as calculated from wheel or tailshaft speed.";
BA_DEF_ SG_  "SPN" INT 0 524287;
BA_DEF_ BO_  "VFrameFormat" ENUM  "StandardCAN","ExtendedCAN","reserved","J1939PG";
BA_DEF_  "BusType" STRING ;
BA_DEF_  "ProtocolType" STRING ;
BA_DEF_DEF_  "SPN" 0;
BA_DEF_DEF_  "VFrameFormat" "J1939PG";
BA_DEF_DEF_  "BusType" "";
BA_DEF_DEF_  "ProtocolType" "";
BA_ "ProtocolType" "J1939";
BA_ "BusType" "CAN";
BA_ "VFrameFormat" BO_ 2364540158 3;
BA_ "VFrameFormat" BO_ 2566844926 3;
BA_ "SPN" SG_ 2364540158 EngineSpeed 190;
BA_ "SPN" SG_ 2566844926 WheelBasedVehicleSpeed 84;
```

Fig. 3. Example of a CAN DBC file

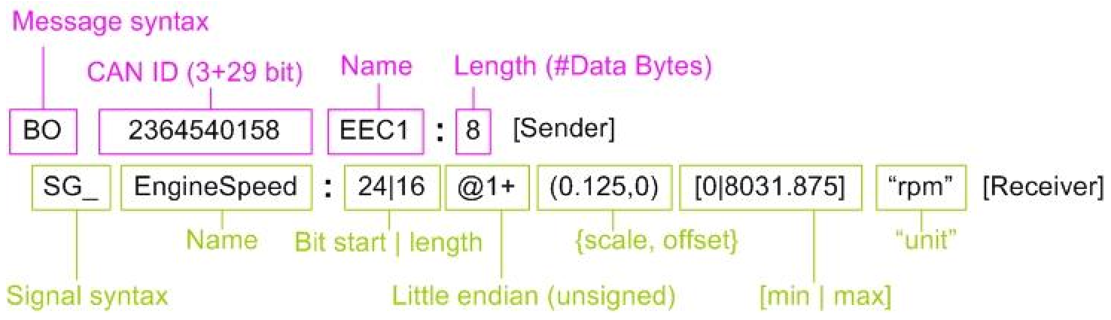


Fig. 4. An example of a message

DBC signal syntax explained:

– each message includes at least 1 signal, at the beginning of which SG_ is indicated;

– the name must be unique, a number from 1 to 32 characters and may also contain letters of the Latin alphabet [A-z], numbers, and an underscore;

– the beginning of the bit is counted from 0 and marks the beginning of the signal in the data payload;

– bit length is the length of the signal;

– @1 indicates that the byte order is little-endian/Intel (compared to @0 for big-endian/Motorola);

– the + symbol indicates that the value type is unsigned (compared to – for signed signals);

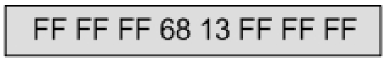
– values (scale, offset) are used in the linear equation of the physical value (more details below);

– [min|max] and units of measurement are additional meta-information that can be omitted;

– the receiver is the name of the recipient node (default is Vector_XXX).

Practice shows that a significant portion of raw CAN data log files contains 20-80 unique controller area network identifiers. Thus, at the first stage, each CAN identifier is mapped to the corresponding DBC conversion rules. For regular 11-bit CAN IDs, this can be done by mapping the CAN ID decimal value to the CAN DBC IDs. For extended 29-bit CAN IDs, a mask (0x1FFFFFFF) must be applied to the 32-bit DBC ID to obtain a 29-bit CAN ID that can then be mapped to the log file [15].

The next step is to set the start, end, and length of the DBC bit to get the corresponding bits from the CAN frame data payload. In this example (Fig. 4), the initial bit is 24 (when counting from 0, it corresponds to 3 bytes), and the bit length is 16 (2 bytes):



FF FF FF 68 13 FF FF FF

Fig. 5. An example of a signal payload

In this example, the "start of bit" in the CAN database file represents the position of the least significant bit (LSB), i.e. little-endian is operated. When adding a database signal to a program to analyze DBC files (eg Vector CANDB++)++), the LSB is also utilized as the start of a bit in the DBC editor. If we instead add a big-endian signal to the DBC's editor user interface, we still will be able to see the LSB as the start of the bit, but when we save the DBC file, the start of the bit is set to the most significant bit (MSB) in the signal. This approach, on the one hand, simplifies the editing of the graphical interface, making it more intuitive, but on the other hand, switching between the graphical interface and the text editor can be a bit confusing. The EngineSpeed signal has a low end (@1), so we need to reorder the byte sequence from 6813 to 1368.

Then we convert the hexadecimal string to decimal and apply a linear transformation:

physical value = offset + scale * original decimal value

621 rpm = 0 + 0.125 * 4968

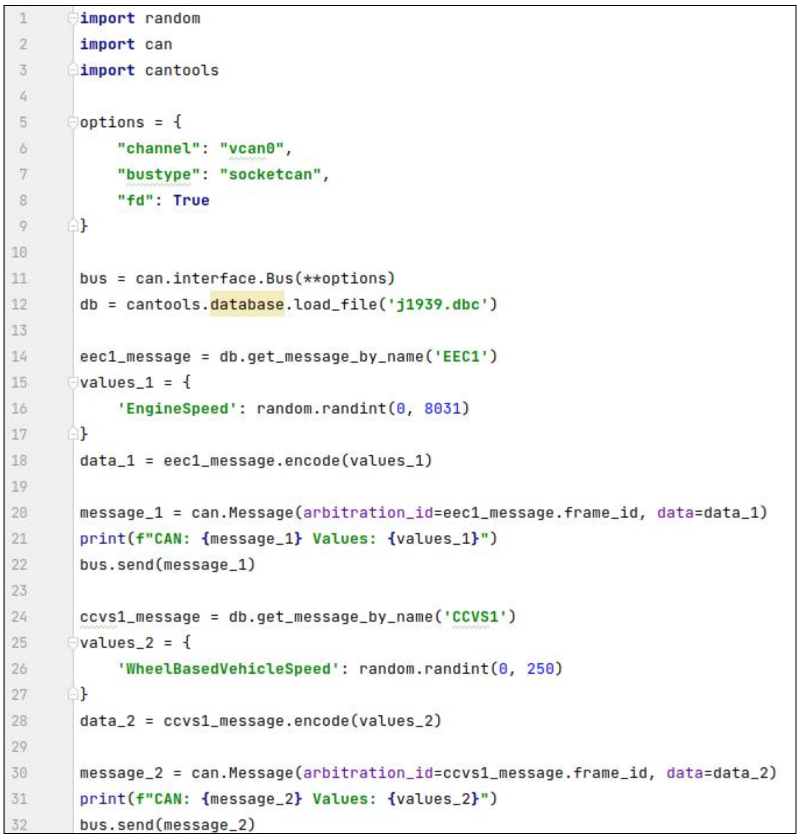
Thus, the physical value of EngineSpeed (also known as scaled engineering value) is 621 rpm.

6. The development of an experimental virtual controller area network

The Linux operating system was operated for the test virtual CAN network since the essential software tools provide an opportunity to create a virtual interface without installing additional programs. Next, a Python script was written for sending coded physical values to the virtual network (Fig. 6).

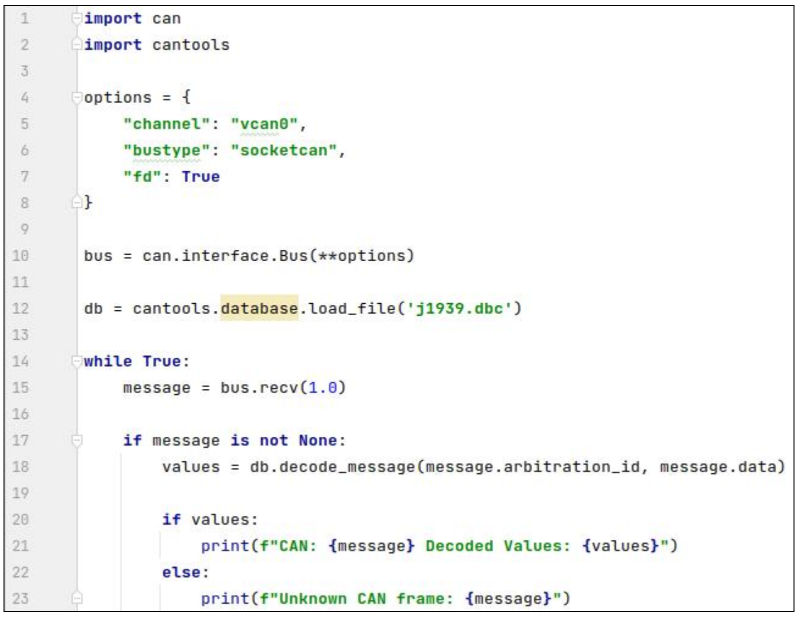
As shown in fig. 7 program is implemented by using the python-can library [16], which provides controller area network support for Python by providing common abstractions for various hardware devices and a set of utilities for sending and receiving messages over the CAN bus. To encode "physical values," we are utilizing an actual DBC file (Fig. 4). We are sending two packets to the network, one containing a random value from 0 to 8031 – this is the value of the engine revolutions. And another message is the speed of the car in the range from 0 to 250. Another script looks much more straightforward since its task is only to decode frames into "physical data" (Fig. 7).

After running the written scripts, we receive the following result (Fig. 8). In the upper part, we can see the result of decoding the CAN packet into "physical values", and in the lower part, the values that are sent to the network. Also, a program such as Wireshark (Fig. 10) [17] can analyze frames.



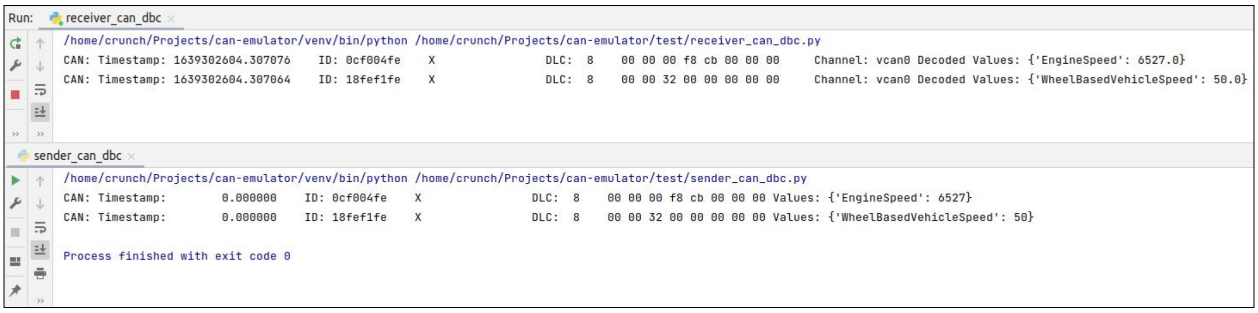
```python
import random
import can
import cantools

options = {
    "channel": "vcan0",
    "bustype": "socketcan",
    "fd": True
}

bus = can.interface.Bus(**options)
db = cantools.database.load_file('j1939.dbc')

eec1_message = db.get_message_by_name('EEC1')
values_1 = {
    'EngineSpeed': random.randint(0, 8031)
}
data_1 = eec1_message.encode(values_1)

message_1 = can.Message(arbitration_id=eec1_message.frame_id, data=data_1)
print(f"CAN: {message_1} Values: {values_1}")
bus.send(message_1)

ccvs1_message = db.get_message_by_name('CCVS1')
values_2 = {
    'WheelBasedVehicleSpeed': random.randint(0, 250)
}
data_2 = ccvs1_message.encode(values_2)

message_2 = can.Message(arbitration_id=ccvs1_message.frame_id, data=data_2)
print(f"CAN: {message_2} Values: {values_2}")
bus.send(message_2)
```

Fig. 6. The script that sends the message



```python
import can
import cantools

options = {
    "channel": "vcan0",
    "bustype": "socketcan",
    "fd": True
}

bus = can.interface.Bus(**options)

db = cantools.database.load_file('j1939.dbc')

while True:
    message = bus.recv(1.0)

    if message is not None:
        values = db.decode_message(message.arbitration_id, message.data)

        if values:
            print(f"CAN: {message} Decoded Values: {values}")
        else:
            print(f"Unknown CAN frame: {message}")
```

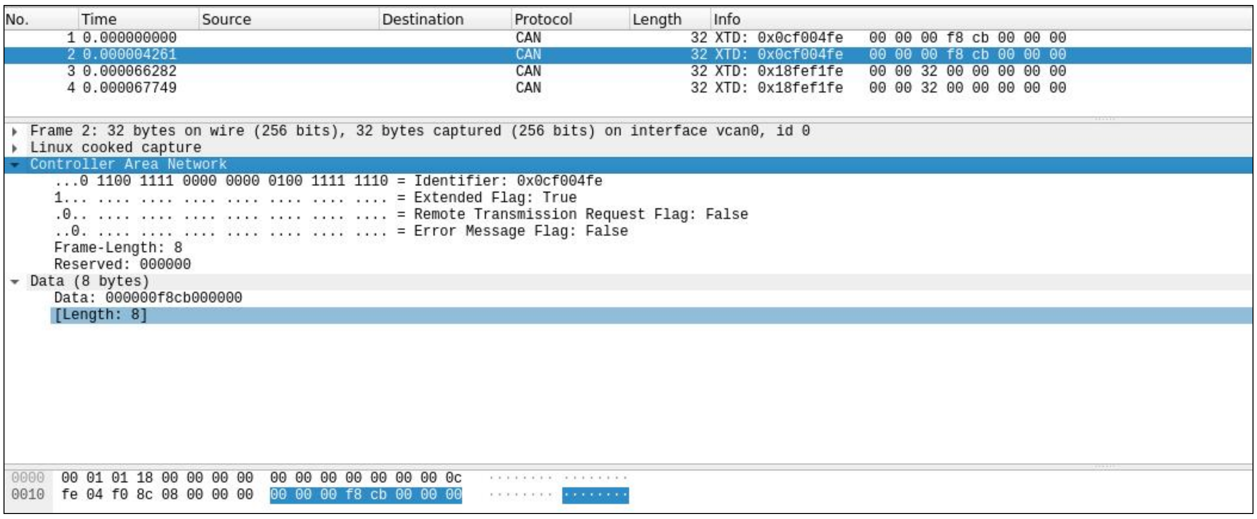
Fig. 7. The script that receives messages



```
/home/crunch/Projects/can-emulator/venv/bin/python /home/crunch/Projects/can-emulator/test/receiver_can_dbc.py
CAN: Timestamp: 1639302604.307076    ID: 0cf004fe    X               DLC:  8    00 00 00 f8 cb 00 00 00     Channel: vcan0 Decoded Values: {'EngineSpeed': 6527.0}
CAN: Timestamp: 1639302604.307064    ID: 18fef1fe    X               DLC:  8    00 00 32 00 00 00 00 00     Channel: vcan0 Decoded Values: {'WheelBasedVehicleSpeed': 50.0}
```

```
/home/crunch/Projects/can-emulator/venv/bin/python /home/crunch/Projects/can-emulator/test/sender_can_dbc.py
CAN: Timestamp:        0.000000    ID: 0cf004fe    X               DLC:  8    00 00 00 f8 cb 00 00 00 Values: {'EngineSpeed': 6527}
CAN: Timestamp:        0.000000    ID: 18fef1fe    X               DLC:  8    00 00 32 00 00 00 00 00 Values: {'WheelBasedVehicleSpeed': 50}

Process finished with exit code 0
```

Fig. 8. The result of encoding and decoding



```
No.     Time         Source     Destination     Protocol    Length  Info
      1 0.000000000                             CAN             32 XTD: 0x0cf004fe   00 00 00 f8 cb 00 00 00
      2 0.000004261                             CAN             32 XTD: 0x0cf004fe   00 00 00 f8 cb 00 00 00
      3 0.000066282                             CAN             32 XTD: 0x18fef1fe   00 00 32 00 00 00 00 00
      4 0.000067749                             CAN             32 XTD: 0x18fef1fe   00 00 32 00 00 00 00 00

▸ Frame 2: 32 bytes on wire (256 bits), 32 bytes captured (256 bits) on interface vcan0, id 0
▸ Linux cooked capture
▾ Controller Area Network
    ...0 1100 1111 0000 0000 0100 1111 1110 = Identifier: 0x0cf004fe
    1... .... .... .... .... .... .... .... = Extended Flag: True
    .0.. .... .... .... .... .... .... .... = Remote Transmission Request Flag: False
    ..0. .... .... .... .... .... .... .... = Error Message Flag: False
    Frame-Length: 8
    Reserved: 000000
▾ Data (8 bytes)
    Data: 000000f8cb000000
    [Length: 8]

0000  00 01 01 18 00 00 00 00  00 00 00 00 00 00 00 0c   ........ ........
0010  fe 04 f0 8c 08 00 00 00  00 00 00 f8 cb 00 00 00   ........ ........
```

Fig. 9. CAN frame in the Wireshark program

Thus, the virtual network created by the authors allows working at the CAN protocol level, does not contain redundant functionality, and can be used for debugging and testing both the whole system and a separate node of the system.

7. Conclusions

Based on the Linux operating system, an experimental virtual controller area network has been developed, which operates with the created software to encode "physical values" and decode them by applying a DBC file. The proposed network has the advantage of a simple user interface and can be availed for debugging and testing not only the entire system but also its nodes.

8. Gratitude

The authors express their gratitude to the staff of the Department of Computerized Automation Systems of Lviv Polytechnic University for valuable feedback and discussions.

9. Conflict of interest

The authors declare that there is no financial or other potential conflict related to this work.

References

[1] L. Görne, H. Reuss, A. Krätschmer, R. Sauerwald. ”Smart data preprocessing method for remote vehicle diagnostics to increase data compression efficiency”. Automotive and Engine Technology, no. 7, 2022, pp. 307–316. DOI: https://doi.org/10.1007/s41104-022-00113-9

[2] M. Di Natale, H. Zeng, P. Giusto, A. Ghosal, Understanding and Using the Controller Area Network Communication Protocol. – New York: Springer, 2012. [Online]. Available: https://books.google.com.py/books?id=rO-EfaSZbMAC&printsec=copyright#v=onepage&q&f=false

[3] A. Ziebinski, R. Cupek, M. Drewniak. “Ethernet-based test stand for a CAN network“.AIP Conf. Proc. 2017, 1906, 120005; DOI: https://doi.org/10.1063/1.5012397

[4] A Mutter. “CAN XL error detection capabilities”. CAN Newsletter no. 2, 2020, pp. 4–12. https://copperhilltech.com/content/CiA%20CAN%20Newsletter%20-%20CAN%20XL%20error%20detection%20capabilities.pdf

[5] Magnus Hell. The physical layer in the CAN XL world, iCC 2021 (international CAN conference). DOI:10.13140/RG.2.2.23239.01448

[6] Basics of the CAN Protocol, 2022. [Online]. Available: https://www.keyence.com/ss/products/daq/lab/candata/protocol.jsp

[7] International standard ISO 11898-1. Road vehicles – Controller area network (CAN). Part 1: Data link layer and physical signaling, 2022. [Online]. Available: https://www.sis.se/api/document/preview/919965/

[8] International standard ISO 11898-2. Road vehicles – Controller area network (CAN). Part 2: High-speed medium access unit, 2022. [Online]. Available: https://www.sis.se/api/document/preview/921358/

[9] H. Zeltwanger, "CAN FD Network Design Hints and Recommendations," SAE Int. J. Passeng. Cars – Electron. Electr. Syst. 9(1):89-92, 2016, DOI: https://doi.org/10.4271/2016-01-0060.

[10] Introduction to the Local Interconnect Network (LIN) Bus, 2022. [Online]. Available: https://www.ni.com/en-us/innovations/white-papers/09/ introduction-to-the-local-interconnect-network--lin--bus.html

[11] Automotive Ethernet: The Future of In-Vehicle Networking, 2022. [Online]. Available: https://blogs.keysight.com/blogs/tech/sim-des.entry.html/2021/06/10/ automotive_ethernet-E6FB.html

[12] DBC Introduction, Open Vehicles, 2020. [Online]. Available: https://docs.openvehicles.com

[13] W Vass, A Comprehensible Guide to J1939. Copperhill Technologies Corporation, 2008.

[14] Understanding CAN DBC, Influx Technology, 2021. [Online]. Available: https://www.influxtechnology.com/post/understanding-can-dbc

[15] An Introduction to J1939 and DBC files, Bryan Hennessy, 2019. [Online]. Available: https://www.kvaser.com/developer-blog/an-introduction-j1939-and-dbc-files/

[16] Python-can library documentation, 2022. [Online]. Available: https://python-can.readthedocs.io/en/master/

[17] Wireshark network protocol analyzer, 2022. [Online]. Available: https://www.wireshark.org/