КОДУВАННЯ ТА ДЕКОДУВАННЯ КАДРІВ ЛОКАЛЬНОЇ МЕРЕЖІ КОНТРОЛЕРІВ ЗА ДОПОМОГОЮ БАЗИ ДАНИХ CAN

Іванюк О.О., к.т.н., доцент, Влах-Вигриновська Г.І., к.т.н., доцент, Модла Р.М., к.т.н., доцент, Кулик Н.З., студент;
Національний університет "Львівська політехніка", Україна,
e-mail: olegiv.mail@gmail.com

Анотація

В статті розглянуто особливості побудови CAN (Controller Area Network - локальна мережа контролерів) мережі в автомобільній галузі. Наведено основні кроки кодування та декодування "фізичних значень" в кадрах CAN та CAN FD ((CAN з гнучкою швидкістю даних)). Проаналізовано синтаксис повідомлень та сигналів у CAN DBC. Розглянуто приклад DBC файлу, який можна використовувати для кодування та декодування швидкості руху та оборотів двигуна вантажного автомобіля. На базі операційної системи Linux та мови програмування python створено експериментальну схему віртуальної CAN-мережі, що на одному вузлі кодує дані, а на іншому декодує з використанням CAN DBC.

Ключові слова

Локальна мережа контролерів, кодування кадрів, декодування кадрів, віртуальна мережа

1. Вступ

Сучасні системи керування та бортової діагностики автомобілів - це складні системи з великою кількістю давачів, виконавчих механізмів та блоків управління, що забезпечують безпеку руху на дорозі за допомогою аналізу отриманих даних та приймання необхідних рішень [1].
На сьогоднішній день в даній галузі застосовуються різні технології зв'язку для побудови локальної мережі між компонентами сучасного автомобіля. Яскравим прикладом може бути CAN протокол, що забезпечує високу надійність та майже необмежену кількість вузлів в мережі [2]. Водночас, незважаючи на технічні характеристики даного протоколу існує велика кількість протоколів вищого рівня, що ускладнюють стандартизацію даної галузі. Практичні дослідження показують, що при розробці власних локальних мереж контролерів (не тільки в автомобільній галузі) виникають проблеми з відлагодження та тестування вузлів [3].
Разом з тим, неможливо не відзначити тенденції розвитку CAN протоколу, уже анонсували новий стандарт (третє покоління) протоколу локальної мережі контролерів, а саме CAN XL, що ще більше загострить проблему інтеграції та тестування, нових та вже існуючих систем з використанням нового стандарту [4]. Проте, CAN XL повинен ще більше розширити спектр застосування локальної мережі контролерів та забезпечити використання даного протоколу у системах з великим об'ємом даних [5].

2. Недоліки

В ході розвитку транспорту виникло багато систем для дистанційної діагностики та керування транспортними засобами, що використовують CAN протокол. Водночас ці системи переважно пропонують комплексні рішення, що часто містять непотрібний для користувача функціонал, або працюють на вищих рівнях та не дають змогу користувачу або розробнику працювати на рівні CAN протоколу.

3. Мета

Метою даної статті є розроблення експериментальної віртуальної CAN мережі, котра надає змогу відлагодження та тестування, як цілої системи, так і окремих її вузлів

4. Особливості роботи шини CAN

Якщо розглядати автомобіль як тіло людини, то шина CAN - це нервова система, що забезпечує зв'язок. У свою чергу, «вузли» або «електронні блоки управління» (ЕБУ) можна розглядати як частини тіла, з'єднані між собою через шину локальної мережі контролерів.
Отже, що таке ЕБУ? В автомобільній системі шини CAN ЕБУ може бути, наприклад, блок управління двигуном, подушки безпеки, аудіосистема тощо. Сучасний автомобіль може мати до 70 ЕБУ - і кожен з них може мати інформацію, якою потрібно поділитися з іншими частинами мережі.
Шина CAN дозволяє кожному ЕБУ спілкуватися з усіма іншими ЕБУ без складної спеціальної проводки. Зокрема, ECU може готувати та передавати інформацію (наприклад, дані датчика) через шину локальної мережі

контролерів (що складається з двох проводів, CAN low і CAN high). Передані дані приймаються всіма іншими ECU в мережі CAN - і кожен ECU може потім перевірити дані та вирішити, отримувати чи ігнорувати їх.
Більш технічно, мережа контролера описується канальним рівнем і фізичним рівнем. У випадку високошвидкісного CAN ISO 11898-1 описує канальний рівень, тоді як ISO 11898-2 описує фізичний рівень [6, 7, 8]. Роль CAN часто представляється в 7-рівневій моделі OSI.
Фізичний рівень шини CAN визначає такі дані, як типи кабелів, рівні електричного сигналу, вимоги до вузлів, імпеданс кабелю тощо. Наприклад, ISO 11898-2 диктує ряд параметрів, зокрема представлених нижче.
Швидкість передачі даних: вузли CAN повинні підключатися через двопровідну шину зі швидкістю передачі до 1 Мбіт/с (класична CAN) або 5 Мбіт/с (CAN FD). Довжина кабелю: максимальна довжина кабелю CAN повинна бути від 500 метрів (125 кбіт/с) до 40 метрів (1 Мбіт/с). Завершення: шина локальної мережі контролерів повинна бути належним чином завершена за допомогою кінцевого резистора шини CAN 120 Ом на кожному кінці шини.
У контексті мереж для автомобільної автоматизації ми часто стикаємося з різними типами мереж. Нижче ми надаємо дуже короткий опис.
Високошвидкісна шина CAN. Шина CAN використовує стандарт ISO 11898, який є найпопулярнішим стандартом CAN для фізичного рівня. ISO 11898 забезпечує швидкість передачі даних в діапазоні від 40 кбіт/с до 1 Мбіт/с (класична CAN). Стандарт набув широкого поширення в сучасних автомобільних додатках, оскільки дає змогу побудови простої кабельної мережі. Він, крім того, став фундаментом для ряду протоколів вищого рівня, зокрема таких як CANopen, OBD2, NMEA 2000, J1939 тощо. Друге покоління CAN отримало назву CAN FD [9].
Низькошвидкісна шина CAN. Стандарт характеризується швидкістю передачі даних в діапазоні від 40 кбіт/с до 125 кбіт/с і забезпечує роботу шини CAN, навіть у випадку несправності на одному з двох проводів. У цій системі для кожного вузла CAN передбачено власне завершення CAN.
Шина LIN. Це однопровідна послідовна шина, яка є більш дешевим доповненням до мереж CAN. Дешевизна LIN пояснюється тим, що реалізація LIN протоколу є повністю програмною і будується на основі звичайного асинхронного інтерфейсу UART. Шину LIN часто використовують для не основних вузлів автомобіля, наприклад кондиціонера, коректора фар, а також для збору даних з простих давачів (температури, світла) [10].
Автомобільна мережа Ethernet [11]. Інтенсивно впроваджується в автомобільному сегменті для підтримки вимог до високої пропускної здатності системи допомоги водієві ADAS, інформаційно-розважальних систем, камер тощо. Автомобільна мережа Ethernet надає значно вищу швидкість передачі даних порівняно з шиною CAN, проте їй бракує деяких характеристик безпеки та продуктивності Classical CAN і CAN FD. Прогнозують, що в короткостроковій перспективі автомобільний Ethernet, CAN FD і CAN XL будуть використовуватися в нових автомобільних та промислових розробках.

5. Основні аспекти кодування та декодування “фізичних значень” в CAN фрейм

Файл база даних CAN - CAN DBC, має текстовий формат і містить дані необхідні для декодування необробленої інформації шини CAN до «фізичних значень» [12].
Розглянемо термін «необроблені дані CAN» на основі прикладу кадру локальної мережі контролерів з вантажівки (рис. 1):

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

Рис. 1. Необроблені дані CAN

Залежно від стандарту фрейм локальної мережі контролерів може містити різну кількість даних в двох основних полях, а саме в ідентифікаторі повідомлення та в корисному навантажені повідомлення. У прикладі (рис. 1), ми використовуємо стандартний CAN протокол, оскільки довжина даних становить 8 байтів.
Якщо у нас є CAN DBC, який містить правила декодування для CAN ID, ми можемо «витягнути» параметри (сигнали) з байтів даних. Одним із таких сигналів може бути EngineSpeed (рис. 2):

Message	Signal	Value	Unit
EEC1	EngineSpeed	621	rpm

Рис. 2. Декодовані фізичні значення

Для кращого розуміння процесу декодування DBC, проаналізуємо синтаксис DBC і приклади покрокового декодування.
Синтаксис повідомлення та сигналу DBC. Спочатку розглянемо реальний приклад файлу CAN DBC (рис. 3).

```
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;
```

Рис. 3. Приклад CAN DBC файлу

Вище наведено демонстраційний файл J1939 DBC, в якому описуються правила декодування швидкості (км/год) і обертів двигуна (об/хв). Ми можемо з вмісту вихідного файлу сформувати новий текстовий файл, перейменувати його, наприклад, j1939.dbc, і застосовувати його для визначення швидкості та оборотів двигуна вантажівок, тракторів або інших великовантажних транспортних засобів [13]. Файл DBC містить правила (рис. 3) проведення декодування повідомлення та сигналу CAN (рис. 4):

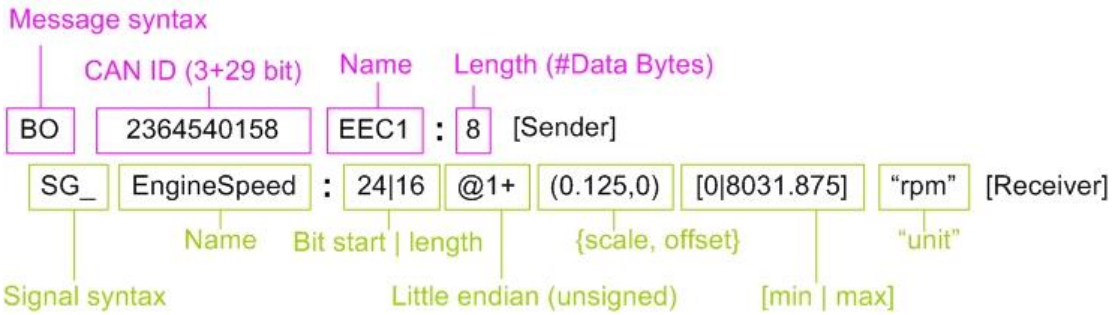


Рис. 4. Приклад повідомлення

Пояснення синтаксису повідомлення DBC [14]:
- на початку повідомлення вказується BO_, а ідентифікатор має бути унікальним і в десятковому (не шістнадцятковому) представленні;
- ідентифікатор DBC додає 3 додаткові біти для 29-бітових ідентифікаторів CAN, які виділяються для прапорця «розширений ідентифікатор»;
- ім'я має бути унікальним, нараховувати від 1 до 32 символів і також може містити літери латинського алфавіту [A-z], цифри та знак нижнього підкреслення;
- довжина (DLC) має бути цілим числом від 0 до 1785;
- назва передавального вузла – відправник, або у випадку якщо ім'я недоступне вказується Vector_XXX.

Пояснення синтаксису сигналу DBC:
- до складу кожного повідомлення входить принаймні 1 сигнал, на початку якого вказується SG_ ;
- ім'я має бути унікальним, нараховувати від 1 до 32 символів і також може містити літери латинського алфавіту [A-z], цифри та знак нижнього підкреслення;
- початок біта відраховується від 0 і позначає початок сигналу в корисному навантаженні даних;
- довжина біта - це довжина сигналу;
- @1 вказує, що порядок байтів є little-endian/Intel (порівняно з @0 для big-endian/Motorola);

- символ + повідомляє, що тип значення - беззнаковий (порівнянні з - для сигналів зі знаком);
- значення (scale, offset) використовуються в лінійному рівнянні фізичного значення (докладніше нижче);
- [min|max] та одиниці вимірювання є додатковою мета-інформацією, яку можна не вказувати;
- приймач - це ім'я вузла одержувача (за замовчуванням вказується Vector_XXX).

Практика показує, що значна частина необроблених файлів журналу даних CAN містить 20-80 унікальних ідентифікаторів CAN. Таким чином, на першому етапі відбувається зіставлення кожного ідентифікатора CAN з відповідними правилами перетворення в DBC. Для звичайних 11-розрядних ідентифікаторів CAN це можна здійснити шляхом зіставлення десяткового значення CAN ID з ідентифікаторами CAN DBC. Для розширених 29-розрядних ідентифікаторів CAN потрібно застосувати маску (0x1FFFFFFF) до 32-розрядного ідентифікатора DBC, щоб отримати 29-розрядний ідентифікатор CAN, який потім можна зіставити з файлом журналу [15].

На наступному етапі необхідно застосовувати початок, кінець та довжину біта DBC, щоб отримати відповідні біти з корисного навантаження даних кадру CAN. У цьому прикладі (рис. 4) початковий біт дорівнює 24 (при підрахунку починаючи з 0 це відповідає 3-ьом байтам), а довжина біта дорівнює 16 (2 байти):

Рис. 5. Приклад корисного навантаження сигналу

У наведеному прикладі «початок біта» у файлі бази даних CAN відображає положення найменшого значущого біта (LSB), тобто застосовується little-endian. У разі додавання сигналу бази даних в програму для аналізу файлів DBC (до прикладу Vector CANDB++), LSB також використовується як початок біту в редакторі DBC. У випадку, коли замість цього до користувацького інтерфейсу редактора DBC додати big-endian сигнал, ми і в цьому разі як початок біту побачимо LSB, проте коли ми збережемо файл DBC, початок розряду встановлюється на старший біт (MSB) у сигналі. Цей підхід, з одного боку, спрощує редагування графічного інтерфейсу, роблячи його більш інтуїтивним, але з іншого боку, при переключенні між графічним інтерфейсом і текстовим редактором може бути дещо заплутаним. Сигнал EngineSpeed має низький кінець (@1), тому нам потрібно змінити порядок послідовності байтів з 6813 до 1368.

Потім ми перетворюємо шістнадцятковий рядок у десятковий і застосовуємо лінійне перетворення:

фізичне значення = зміщення + масштаб * вихідне десяткове значення

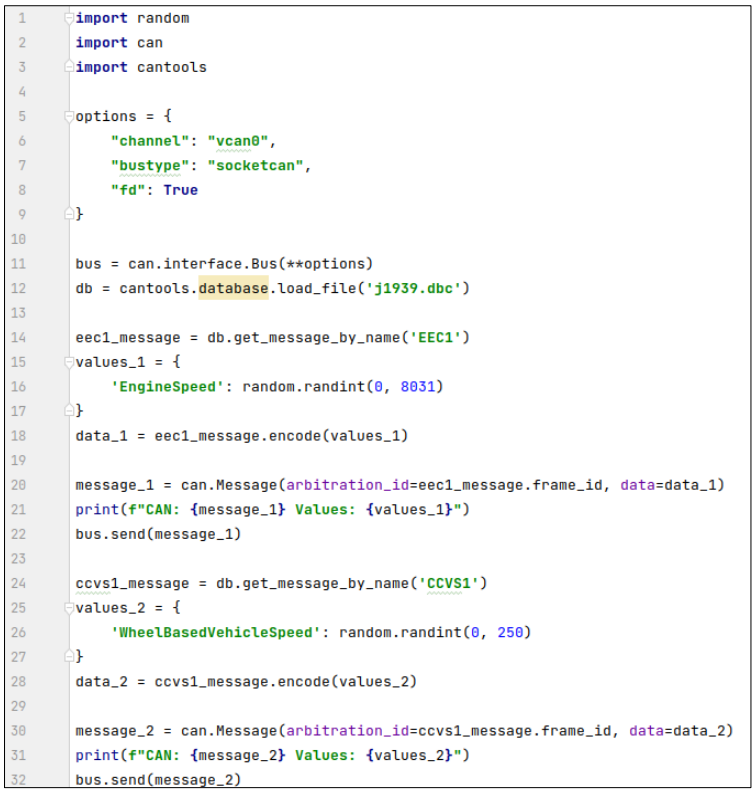
621 об/хв = 0 + 0,125 * 4968

Таким чином, фізичне значення EngineSpeed (воно ж масштабоване інженерне значення) становить 621 об/хв.

6. Розроблення експериментальної віртуальної CAN мережі

Для тестової віртуальної CAN мережі використовувалася операційна система Linux, оскільки базові програмні засоби надають можливість створити віртуальний інтерфейс без встановлення додаткових програм.

Далі було написано пайтон скрипт для надсилання кодованих фізичних значень в віртуальну мережу (рис. 6).



```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)
```

Рис. 6. Скрипт, що надсилає повідомлення

```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}")
```

Рис. 7. Скрипт, що отримує повідомлення

Наведена на рис. 6 програма реалізована за допомогою бібліотеки python-can [16], яка забезпечує підтримку CAN для Python, надаючи загальні абстракції для різних апаратних пристроїв і набір утиліт для надсилання та отримання повідомлень по шині CAN. Для кодування “фізичних значень” ми використаємо реальний DBC файл (рис. 3). В мережу надішлемо два пакети, один буде містити випадкове значення від 0 до 8031 - це буде значення оборотів двигуна. А інше повідомлення - це швидкість автомобіля в діапазоні від 0 до 250. Інший скрипт, виглядатиме значно простіше, оскільки його завдання полягає тільки в декодуванні кадрів в “фізичні дані” (рис. 7).

Після запуску написаних скриптів, ми отримаємо наступний результат (рис. 8). У верхній частині ми можемо побачити результат декодування CAN пакетів в “фізичні значення”, а у нижній значення, які надсилаються в мережу. Також, для аналізу фреймів можна використати, таку програму як Wireshark (рис. 9) [17].

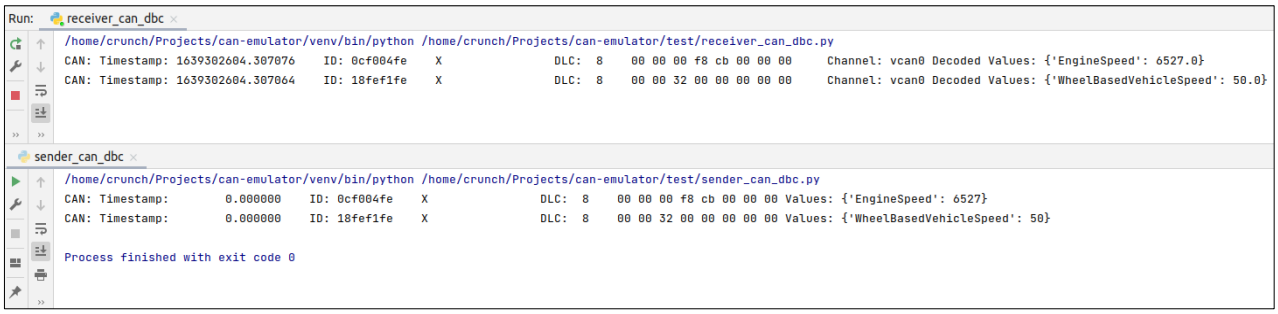


Рис. 8. Результат кодування та декодування

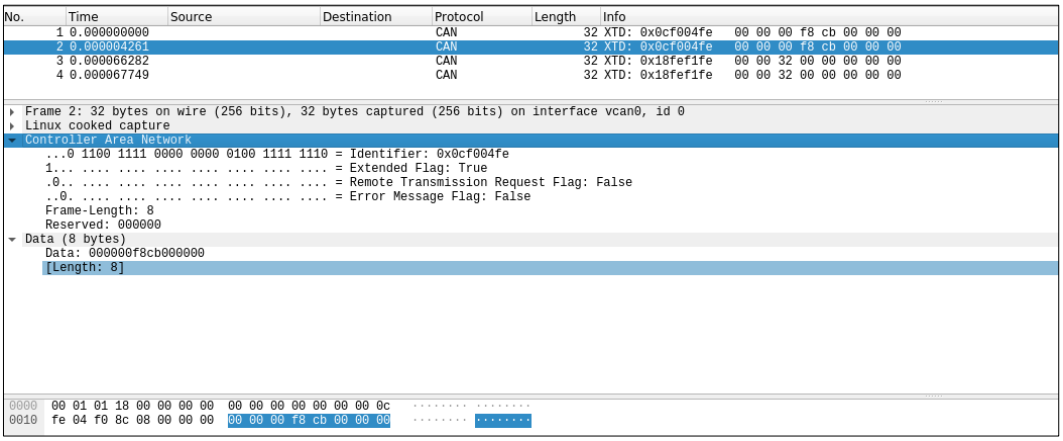


Рис. 9. CAN фрейм у програмі Wireshark

Таким чином, створена авторами віртуальна мережа дає змогу працювати на рівні CAN протоколу, не містить надлишкового функціоналу і може бути використана для відлагодження та тестування, як цілої системи, так і окремого вузла системи.

7. Висновки

На основі операційної системи Linux розроблено експериментальну віртуальну CAN мережу, яка за допомогою створеного програмного забезпечення виконує кодування “фізичних значень” та їх декодування з використанням DBC файлу. Запропонована CAN мережа має перевагу простого користувацького інтерфейсу і може бути використана для відлагодження та тестування не лише цілої системи, але й окремих її вузлів.

8. Подяка

Автори висловлюють подяку співробітникам кафедри комп’ютеризованих систем автоматизації НУ «Львівська політехніка» за цінні відгуки та обговорення.

9. Конфлікт інтересів

Автори заявляють про відсутність будь-якого фінансового або іншого можливого конфлікту, який стосується цієї роботи.

Посилання

[1] L. Görne, H. Reuss, A. Krätschmer, R. Sauerwald. ”Smart data preprocessing method for remote vehicle diagnostics to increase data compression efciency”. 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 signalling, 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/