Electronic Control Units for Electric Vehicles

CALIN CIUFUDEAN, CORNELIU BUZDUGA
Computers, Electronics and Automation,
Stefan cel Mare University,
13 University str., 720229, Suceava,
ROMANIA

Abstract: - This article discusses the development of a prototype electronic control unit for electric vehicles developed in our discrete event systems laboratory. This system aims to enhance the tools for test and diagnostics based on the controller area network for the automotive industry, mainly for electric vehicles (EV) to ensure affordable integration on the market. The project's primary objective is to create a financially affordable solution, i.e., lowering production costs by introducing tests and a diagnostic environment similar to the electronic control units. This will finally deliver non-prohibitively expensive EVs for individual consumers.

Key-Words: - Automotive industry, electric vehicle, microcontroller, automated system, Unified Diagnostic Services, Controller Area Network.

Received: April 16, 2024. Revised: November 11, 2024. Accepted: December 9, 2024. Published: December 31, 2024.

1 Introduction

This paper describes the development of a communication simulator using the Controller Area Network (CAN) bus and the Unified Diagnostic Services (UDS) standard. The main goal of the work is to create a test and diagnostic environment similar to the electronic control units (ECUs) used in the automotive industry. A development kit on Infineon Tricore TC297 was used for the implementation. The communication part was done using the C programming language, and the software CAN UDS Simulator, which was developed in C# using Windows Forms, was used for the communication part.

Because the Infineon Tricore TC297 microcontroller has high performance and native CAN support, it was chosen to implement CAN communication, [1], [2], [3], [4]. C source code offers efficiency and direct control over hardware. The CAN UDS Simulator software provides a user-friendly testing interface that configures and sends UDS commands to the simulator, making it easy to test and diagnose ECU functionality straightforwardly. As automotive technology rapidly advances, the increasing complexity of communication networks and control systems becomes crucial for the operation of modern vehicles. The Controller Area Network (CAN) is widely utilized in the automotive industry due to its efficiency and reliability. It enables Electronic Control Units (ECUs) to communicate with one another, coordinating essential vehicle functions such as braking, steering, and engine control.

Numerous benefits can be obtained by using our simulator, including the following:

- Students can develop the skills necessary for the automotive industry by applying theoretical knowledge to real-world situations.
- Safety: the ability to perform tests and failure scenarios in a safe and controlled environment without facing the risks associated with testing on real vehicles
- Flexibility: the ability to configure and test a wide range of communication and diagnostic scenarios without the limitations of real hardware provides students with a wide range of learning experiences.

The remainder of this paper, section II discusses the system architecture and section III focuses on the software support of the CAN UDS Simulator. Section IV concludes our work and suggests further possible development of it.

2 The System Architecture

The system comprises the Infineon Tricore TC297-based development kit and a USB to CAN FD Automotive USB interface from Ixxat. Communication starts at the TC297 microcontroller level, which, through the CAN driver, prepares the datagram to be transmitted by the TLE7250GVIO integrated circuit, which translates from the CAN protocol level to the bus level, [5], [6], [7].

The datagram is sent on the bus to the USB interface, which receives and transmits data from an easy-to-use interface on the Windows operating system (Figure 1).

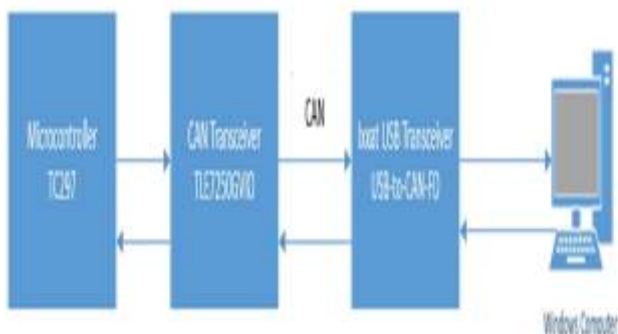

Fig. 1: The architecture of the developed system

The Unified Diagnostic Services (UDS) diagnostic protocol, widely used in the automotive industry, is designed to communicate and manage diagnostic information between test devices and electronic control units (ECUs). The ISO 14229 specification describes the UDS standard and establishes the structure and functionality of diagnostic messages transmitted over communication buses such as Ethernet, CAN (Controller Area Network), LIN, and FlexRay, [8], [9], [10].

UDS possesses multiple diagnostics departments specialized in performing certain tasks. Each service may have a unique code, which may have additional parameters to define diagnostic „operations” and is referred to as service ID or service code.

Unified services diagnostics are necessary for almost all modern vehicle service and repair operations. The main UDS activities include, [10], [11], [12].

Diagnostics and Repair: Reading and clearing Diagnostic Trouble Codes (DTCs) helps in pinpointing and correcting technical problems of the vehicles.

- Firmware Update: In this section, updates may be loaded, and also downloaded from the ECU.
- Systems Setup: This function helps in setting up and re-calibrating some of the vehicle components for proper operation.
- Performance Monitoring: This function allows retrieval of performance information and the state of the vehicle systems for evaluation and analysis.

Each group of functions solves a set of maintenance or diagnostics problems about the vehicles. It provides an integrated solution for communication management and ECU control in modern vehicles. These are the basic functions important for the problem-free operation of the vehicle and software maintenance, and fast, precise, and effective diagnostics.

The structure of a UDS datagram is accompanied by the following points:

- UDS Service ID (SID): The first octet of the datagram is the UDS Service ID (SID) which indicates the type of service request.

CAN request a message (Table 1):

Table 1. CAN request message.

Message ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
0x101	0x3E	0x0	0x0	0x0	0x0	0x0	0x0	0x0

CAN response message (Table 2):

Table 2. CAN response message.

Message ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
0x102	0x7E	0x0	0x0	0x0	0x0	0x0	0x0	0x0

CAN request message (Table 3):

Table 3. CAN request message.

Message ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
0x101	0x2E	0xB5	Buzz	0x00	0x0	0x0	0x0	0x0

CAN possible response messages (Table 4 and Table 5)

Table 4. Positive answer.

Message ID	Data0	Data1	Data2	Data3	Data4	Data5	Data6	Data7
0x102	0x6E	0xB5	0x0	0x0	0x0	0x0	0x0	0x0

Table 5. Negative answer

Message ID	Data 0	Data 1	Data 2	Data 3	Data 4	Data5	Data6	Data7
0x102	0x7F	0x2E	0xB5	Err	0x0	0x0	0x0	0x0

3 Development of the CAN UDS Simulator Software Application

The "CAN UDS Simulator" application was developed to automate the testing functions implemented in the embedded system application. This application was made in the C# programming language, using the Ixxat development library.

It is a robust and reliable technology for developers who want to create Windows desktop applications with short development time and extensive functionality (Figure 2).

```
0 references
static void Main()
{
    canAdapter canAdapter = new canAdapter();
    canAdapter.SelectDevice();
    canAdapter.InitSocket(0);
    canAdapter.StartRxThread();
    ApplicationConfiguration.Initialize();
    Form1 form1 = new Form1();
    form1.canAdapter = canAdapter;
    Application.Run(form1);
    canAdapter.SetQuitFlagReceivedThread();
    canAdapter.JoinRxThread();
    canAdapter.FinalizeApp();
}
```

Fig. 2: The main function of the CAN UDS Simulator app

In the application's main function, a class necessary for using the CAN interface is instantiated for the first time. Next are the functions for selecting the device, initializing the communication channel, and starting a thread to receive datagrams. After that, the application configuration is initialized, the class representing the visual interface is instantiated, the CAN interface is passed, and the visual application starts running. The last three functions are required to complete the running so that resources are reallocated, and no manual intervention is required.

For the best datagram processing time, a parallel thread has been implemented. This thread runs in the background and calls the message reading function when a receive event is detected (Figure 3).

```
1 reference
public void StartRxThread()
{
    rxThread = new Thread(new ThreadStart(ReceiveThreadFunc));
    rxThread.Start();
}
```

Fig. 3: Parallel thread start function for receiving CAN datagrams

When a message is successfully received, the routine to read the CAN message from the USB interface receive stack is called. The validated message is passed to the message verification routine. Three additional routines are required to complete the application run. First, the variable that closes the execution of the receiving parallel thread must be set. Setting this variable stops the while loop in Figure 4.

```
1 reference
public void SetQuitFlagReceivedThread()
{
    Interlocked.Exchange(ref mMustQuit, 1);
}
```

Fig. 4: Function to set parallel thread stop variable

Afterwards, the execution thread must be closed. This operation is done by the "Join()" method specific to the class that describes the threads. This closes the instantiated objects for reading and writing messages and uninitialized the CAN channel, the CAN controller, and the USB interface (Figure 5).

```
1 reference
public void JoinRxThread()
{
    rxThread.Join();
}
```

Fig. 5: The function that closes the thread of execution

Creating an automated test requires two aspects: the front end and the back end. The front end involves adding a label with the test name, a test box for the parameters, a submit button, and a label for the result, which is updated by the back end with the test result, pass or fail (Figure 6 and Figure 7).

Fig. 6: Launching the CAN UDS Simulator application

The back-end part of the "Tester Present" test is described next.

```
1 reference
private void button1_Click(object sender, EventArgs e)
{
    canAdapter.TransmitTesterPresent();
    Thread.Sleep(101);
    if (canAdapter.GetTesterPresentReceived() == true)
    {
        label2.Text = "Passed";
        label2.ForeColor = Color.Green;
    }
    else
    {
        label2.Text = "Failed";
        label2.ForeColor = Color.Red;
    }
    canAdapter.SetTesterPresentReceived(false);
    canAdapter.SetTesterPresentSent(false);
}
```

Fig. 7: The source code behind the "Send" command

When the "Send" button is pressed, the request message is sent over the CAN to the embedded system, then it waits 101 milliseconds to receive the response (Figure 8).

```
1 reference
public void TransmitTesterPresent()
{
    IMessageFactory factory = VciServer.Instance().MsgFactory;
    ICanMessage canMsg = (ICanMessage)factory.CreateMsg(typeof(ICanMessage));

    canMsg.TimeStamp = 0;
    canMsg.Identifier = 0x101;
    canMsg.FrameType = CanMsgFrameType.Data;
    canMsg.DataLength = 8;
    canMsg.SelfReceptionRequest = true;

    for (Byte i = 0; i < canMsg.DataLength; i++)
    {
        canMsg[i] = 0;
    }

    canMsg[0] = 0x3e;

    mWriter.SendMessage(canMsg);

    TesterPresentSent = true;
}
```

Fig. 8: The function of transmitting a message on CAN

The function of sending a message on CAN is specific to each test. A message is instantiated and configured with a time stamp, identifier, frame type, and data length. The message is then populated with the data to be transmitted and sent. The send confirmation variable for the test is set, then the program continues its execution (Figure 9).

```
1 reference
static void CheckAnswer(ICanMessage canMessage)
{
    if(canMessage.Identifier != 0xffffffff)
    {
        if (canMessage[0] == 0x7E)
        {
            TesterPresentReceived = true;
        }
    }
}
```

Fig. 9: Verification of receipt of the "Tester Present" message

Through the parallel thread, when the message is successfully received, it is checked to see if it matches the expected one. If the validation is positive, the result variable is set with the value "true" and otherwise with false. Through a "getter" type method, the test result of the CAN adapter class is obtained. If the result is positive, the text "Passed" written in green will be displayed next to the text. If it fails, it will show "Failed" written in red (Figure 10 and Figure 11).

The "setter" and "getter" type methods, specific to object-oriented programming, ensure the communication between the CAN interface class and the visual application class. For each test, two static variables are created in the CAN interface class, each having one method for setting, and one for obtaining, with the role of being accessed from other classes (Figure 12).

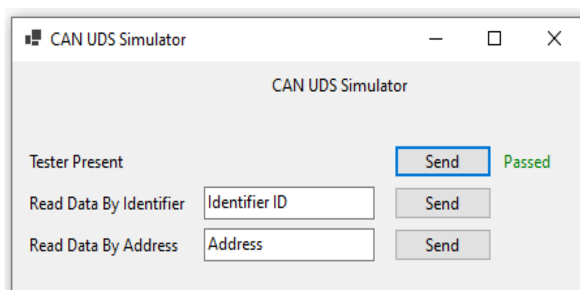

Fig. 10: Positive result for the "Tester Present" test.

Fig. 11: Negative results for the "Tester Present" test

```
static bool TesterPresentSent;
static bool TesterPresentReceived;


0 references
public bool GetTesterPresentSent()
{
    return TesterPresentSent;
}

1 reference
public bool GetTesterPresentReceived()
{
    return TesterPresentReceived;
}

1 reference
public void SetTesterPresentSent(bool flag)
{
    TesterPresentSent = flag;
}

1 reference
public void SetTesterPresentReceived(bool flag)
{
    TesterPresentReceived = flag;
}
```

Fig. 12: Variables and methods required for the "Tester Present" test

4 Conclusion

This project required the development of a CAN bus communication simulator that conforms to the UDS standard. This simulator has been implemented on an Infineon Tricore TC297 development kit. This is in line with the research aim of the paper, which was ‘To design and develop a CAN UDS Simulator to capture and interpret data from CAN communication and also to develop command test software to test the data received from the simulator in an efficient and easily interpretable manner as possible. Reason: Enhanced comprehension and clarity through the application of more complex

language structures and increased use of technical terminology. To prove, we have been able to simulate CAN bus communication and also apply the UDS standard in a practical and convenient way. We have developed a strong and flexible application using C programming and the Infineon Tricore TC297 Development Kit. The CAN UDS Simulator software, developed in C# and Windows Forms, is a graphical user interface (GUI) used to test and diagnose different types of CAN communication. Reason: Expanded vocabulary and improved writing with some adjustments made for easier reading and verification of information. This simulator was useful for users to get a feel and have a go in a safe and controlled way without messing up real vehicle testing, thus increasing their knowledge of CAN protocols, UDS standards, and the critical technical skills needed in the automotive industry.

Further developments:

- Implementation of the Ethernet connection, or another communication protocol available on the development kit (CAN-FD, UART, LIN, FlexRay);
- Improving the catalog of messages;
- Microcontroller reprogramming via a communication protocol, without the help of USB miniWiggler JDS.

Declaration of Generative AI and AI-assisted Technologies in the Writing Process

During the preparation of this work the authors used Grammarly in order to improve the readability and language of the manuscript. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

References:

[1] UDS Explained – A Simple Intro, 2024, [Online]. https://www.csselectronics.com/pages/uds-protocol-tutorial-unified-diagnostic-services (Accessed Date: December 12, 2024).

[2] S. Shehryar, H. Jamil, N. Iqbal, S. Khan, Evolving Electric Mobility: In-Depth Analysis of Integrated Electronic Control Unit Development in Electric Vehicles, *IEEE Access,* p.p. (99):1-1, 2024, doi: 10.1109/ACCESS.2024.3356598.

[3] C. Armenta-Deu and Th. Coulaud, Control Unit for Battery Charge Management in Electric Vehicles (EVs), *Future Transp.*, Vol. 4. No. 2, 429-449, 2024.

[4] Tesla's New 12V Li-Ion Auxiliary Battery Has CATL Cells Inside, Mark Kane. *Inside EVs*, 7 November 2021.

[5] Tesla Flat 12 V Battery. Tesla Info. Updated 12 September 2022, [Online]. https://tesla-info.com/blog/tesla-flat-battery.php (Accessed Date: December 15, 2024).

[6] Tesla Confirms the Switch to 48 Volt System, Mark Kane. *Inside EVs*, 11 March 2023, [Online]. https://insideevs.com/news/656775/tesla-switch-48v-voltage-system (Accessed Date: December 10, 2024).

[7] C. Jia, H. He, J. Zhou, J. Li, Z. Wei, K. Li, Learning-based model predictive energy management for fuel cell hybrid electric bus with health-aware control. *Appl. Energy*, Vol. 355, 2024, https://doi.org/10.1016/j.apenergy.2023.122228.

[8] C. Jia., J. Zhou, H. He, J. Li, Z. Wei, K. Li, M. Shi, A novel energy management strategy for hybrid electric bus with fuel cell health and battery thermal and health-constrained awareness. *Energy, 271,* 2023.

[9] H.S. Kim, M.H. Ryu, J.W. Baek, J.H. Jung, High-efficiency isolated bidirectional AC–DC converter for a DC distribution system. *IEEE Trans. Power Electron.*, *28*, 1642–1654, 2012.

[10] M. A. Khan, Adaptive Control Mechanisms for Electric Vehicles: A Study Based on Driving Environments, *JSM Computer Science & Engineering,* 3(1): 1008, 2024.

[11] ECU Testing for Electric and Hybrid Vehicles, [Online]. https://cdn.vector.com/cms/content/know-how/_technical-articles/EMobility_VTSystem_AEL_201108_PressArticle_EN.pdf (Accessed Date: December 12, 2024).

[12] F. Un-Noor, S. Padmanaban, L. Mihet-Popa, M.N. Mollah, A Comprehensive Study of Key Electric Vehicle (EV) Components, Technologies, Challenges, Impacts, and Future Direction of Development, *Energies,* 2017, Vol.10, No. 8, doi: 10.3390/en10081217.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)
The authors equally contributed to the present research, from formulating the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself
No funding was received to conduct this study.

Conflict of Interest
The authors have no conflicts of interest to declare.

Creative Commons Attribution License 4.0 (Attribution 4.0 International, CC BY 4.0)
This article is published under the terms of the Creative Commons Attribution License 4.0
https://creativecommons.org/licenses/by/4.0/deed.en_US