INTRODUCTION

Electronic control units (ECUs) are an integral part of modern vehicles, controlling everything from safety procedures and system diagnostics to engine optimization. The increasing complexity of modern vehicles has led to a need for the development of sophisticated diagnostic systems capable of detecting, identifying, and addressing faults within various ECUs. These ECUs are responsible for controlling key vehicle functionalities such as the engine, transmission, braking systems, and other critical components. As vehicles continue to evolve with more electronics and software-based systems, the need for a standardized and efficient method of communicating with these ECUs has become a major important need. This is where the Unified Diagnostic Services (UDS) protocol, standardized under ISO 14229, plays a vital role.

The UDS protocol is a widely adopted communication protocol in the automotive industry, primarily used for vehicle diagnostics and reprogramming. It enables seamless interaction between a diagnostic tool and the vehicle's onboard systems via a communication bus, typically the CAN (Controller Area Network). UDS provides a standard set of services that can be used to perform various diagnostic functions, such as reading diagnostic trouble codes (DTCs), controlling diagnostic sessions, performing ECU resets, and programming ECUs.

1.1. Understanding the UDS Protocol

Exploring the structure, layers, and services provided by the UDS protocol, including how it enables diagnostic communication over a vehicle network.

1.2. Designing a UDS Communication Stack

Developing the software stack required to implement UDS on an embedded platform, using an STM32 microcontroller or a similar automotive-grade microcontroller.

1.3. Implementing Key UDS Services

Building and testing essential UDS services such as:

- Read Data by Identifier (0x22)
- Write Data by Identifier (0x2E)
- Routine Control (0x31)
- Read Diagnostic Trouble code (0x19)
- Clear Diagnostic Information (0x14)

1.4. Communication with the CAN Bus

Implementing the ISOTP (ISO 157652) protocol for message segmentation and transport over the CAN bus.

This project has practical implications in the field of automotive diagnostics, where accurate and efficient fault detection is critical for vehicle maintenance and repair. By implementing UDS, technicians can query the vehicle's ECUs for detailed information on system status, retrieve error codes, and reset or reprogram the vehicle's electronic components. This helps in significantly reducing the time required for troubleshooting and ensuring the optimal performance of the vehicle.

LITERATURE SURVEY

2.1. Implement Standard UDS Diagnostics Over CAN (2024)

This 2024 paper author U. Kataria, K. Panchal, J. Puvvada, S. Kadlag and S. Shinde focuses on implementing the UDS protocol over the CAN bus in compliance with the ISO 157652 (ISOTP) transport layer protocol. The research demonstrates the development of a robust UDS communication stack tailored for automotive diagnostic environments, using CAN for reliable data transport between ECUs and external diagnostic tools. The study details the integration of UDS services like Request Download (0x34) and Transfer Data (0x36), emphasizing the importance of ISOTP for managing segmented messages in UDS communication. The paper also highlights the significance of UDS in enabling advanced diagnostics in modern vehicles, particularly for software updates and ECU reprogramming over CAN networks. [1]

2.2. Design of UDS Protocol in an Automotive Electronic System (2023)

This 2023 study author M. Kuntoji, V. Medam and Veena Devi S. V, focuses on the design and implementation of the UDS protocol in a fully integrated automotive electronic system. The research explores the role of UDS in managing diagnostics, ECU communication, and fault code retrieval. The authors present a detailed methodology for developing a UDS stack, with emphasis on key diagnostic services such as Read DTC Information (0x19) and Control DTC Setting (0x85). The study also discusses the challenges faced in resource constrained embedded environments, providing recommendations for optimizing UDS performance in modern automotive ECUs. This paper contributes to the understanding of UDS implementation within realworld automotive systems.[2]

2.3. Development of UDS on CAN Using MATLAB and Arduino (2022)

This 2022 paper author S. Desai and Y. Bhateshvar explores the implementation of the UDS protocol on an Arduino platform, using MATLAB to simulate and test the diagnostic services over the CAN bus. The research demonstrates a low-cost approach to understanding UDS by utilizing accessible tools such as Arduino for embedded system design. The authors successfully implemented key UDS services, including Diagnostic Session Control (0x10) and Clear Diagnostic Information (0x14), highlighting the flexibility of UDS for educational and prototyping purposes. This work provides a practical example of UDS implementation in a simplified environment, offering insights into how UDS functions in realtime systems.[3]

2.4. ISO 142291:2013 Unified Diagnostic Services (UDS)

The ISO 142291:2013 document defines the Unified Diagnostic Services (UDS) protocol, which standardizes communication between vehicle Electronic Control Units (ECUs) and external diagnostic tools. UDS is crucial for performing diagnostics, ECU

reprogramming, and fault code management in modern vehicles. This standard defines a comprehensive set of services for vehicle diagnostics, such as Diagnostic Session Control (0x10), ECU Reset (0x11), Read Data by Identifier (0x22), and Clear Diagnostic Information (0x14). UDS operates across various communication channels, including CAN, Ethernet, and FlexRay, enabling diagnostic tools to interact with a wide range of vehicle systems. The document ensures that diagnostic tools and ECUs across different manufacturers can communicate using a unified and standardized approach, promoting interoperability in the automotive industry.[4]

2.5. ISO 11898 Controller Area Network (CAN)

The ISO 11898 standard specifies the Controller Area Network (CAN) protocol, which is widely used for communication between ECUs in automotive systems. CAN enables realtime, lowlatency communication in embedded systems, making it ideal for vehicle applications where reliability and data integrity are critical. The standard defines both the CAN protocol for data link and physical layer specifications. CAN is particularly suited for vehicle diagnostics because it allows multiple ECUs to communicate on the same bus, reducing wiring complexity and ensuring prioritized message handling. The combination of UDS over CAN, known as Diagnostic over CAN (DoCAN), allows diagnostic tools to access ECU data, read diagnostic trouble codes (DTCs), and perform other essential diagnostic functions. [5]

2.6. ISO 157652:2016 ISO Transport Protocol (ISOTP)

The ISO 157652 standard, also known as ISO Transport Protocol (ISOTP), provides a communication framework for sending diagnostic messages over the CAN bus. Since CAN frames have a limited data size (typically 8 bytes), larger diagnostic messages such as UDS requests need to be segmented and reassembled. ISOTP ensures the correct transport of UDS messages by defining a method for segmenting large messages, flow control, and message reassembly. This protocol is essential for UDS over CAN, as it enables diagnostic services to send larger payloads efficiently, ensuring communication reliability and data integrity even with the CAN protocol's bandwidth constraints. ISOTP plays a pivotal role in making UDS functional over CAN, as it handles message fragmentation and ensures that services like Request Download (0x34) and Transfer Data (0x36) can be executed seamlessly. [6]

2.7. ISO 13400 Diagnostic Communication Over Internet Protocol (DoIP)

The ISO 13400 standard specifies Diagnostic Communication over Internet Protocol (DoIP), which allows UDS services to be transmitted over Ethernet instead of traditional automotive communication buses like CAN. DoIP offers higher bandwidth, making it ideal for modern vehicles that require faster and more robust diagnostic communication, particularly for large data transfers and remote diagnostics. Ethernet based diagnostic communication provides significant advantages in terms of speed and scalability compared to CAN, making it suitable for more advanced vehicles that

support highs peed communication, such as electric and autonomous vehicles.[7]

2.8. ISO 26262 Functional Safety

The ISO 26262 standard for Functional Safety in automotive systems emphasizes the importance of reliable diagnostics to ensure that vehicle systems are functioning safely. While this standard is not directly related to UDS or communication protocols like CAN or DoIP, it provides guidelines on ensuring that diagnostic systems, including UDS-based systems, meet stringent safety requirements. Implementing UDS in compliance with ISO 26262 helps ensure that diagnostic communication is reliable, even in safety-critical systems, contributing to the overall functional safety of the vehicle. [8]

PROBLEM STATEMENT

In modern automotive systems, the increasing complexity of vehicle electronics and the integration of numerous Electronic Control Units (ECUs) require robust, standardized diagnostic mechanisms. Vehicles today rely heavily on electronics for critical functions such as engine control, transmission, braking, safety systems, and infotainment. As a result, detecting, diagnosing, and troubleshooting faults in these systems has become increasingly challenging. Traditional diagnostic methods are often manufacturer specific, lack standardization, and require extensive manual intervention, making the process inefficient and error-prone.

The Unified Diagnostic Services (UDS) protocol, defined under ISO 14229, was developed to address these challenges by providing a standardized method for communication between vehicle ECUs and external diagnostic tools. UDS enables functionalities such as reading and clearing fault codes, retrieving system data, and programming ECUs. However, implementing the UDS protocol in an automotive system, particularly over the Controller Area Network (CAN) bus, poses several technical challenges:

3.1. Lack of Standardized Diagnostic Communication in ECUs

Modern vehicles contain multiple ECUs from various manufacturers, each responsible for different subsystems like the engine, transmission, safety, and infotainment. Without a standardized protocol, diagnosing faults across these subsystems is cumbersome, requiring specific tools and protocols for each ECU. This increases maintenance costs and complicates troubleshooting for service technicians.

3.2. Complexity of Implementing UDS in Embedded Systems

Implementing the UDS protocol on a microcontroller or embedded system requires precise handling of message segmentation, transport protocols, and error handling. The CAN bus used in automotive communication has limitations in terms of message size (typically 8 bytes), which necessitates the use of ISOTP (ISO 157652) to segment larger UDS messages. This adds complexity to the UDS stack implementation, as developers must manage the reassembly of fragmented messages and ensure timely delivery over realtime communication networks like CAN.

3.3. Efficient Fault Detection and Troubleshooting

As vehicles become more reliant on electronics, identifying the root cause of a system malfunction is crucial. Traditional diagnostic tools may fail to quickly or accurately pinpoint issues due to the lack of unified diagnostic services across all ECUs. There is a need for a comprehensive diagnostic solution that can detect, log, and clear fault codes in a timely and efficient manner.

3.4. Challenge of ECU Reprogramming

One of the critical applications of UDS is ECU reprogramming, which allows for updating software on ECUs after a vehicle has left the factory. Reprogramming ECUs over the CAN network is challenging due to the constraints of data transmission size and reliability. Ensuring secure and uninterrupted reprogramming over a constrained network environment is a problem that requires careful implementation of diagnostic protocols and error recovery mechanisms.

3.5. RealTime Performance Constraints

In automotive diagnostics, realtime performance is essential. The diagnostic tool must be able to quickly retrieve and respond to fault codes, monitor system parameters, and handle dynamic events within the vehicle. Implementing UDS on an embedded platform with realtime constraints, such as the STM32 microcontroller, requires careful optimization of the software stack to ensure timely communication and processing of diagnostic requests.

3.6. Interoperability Between Diagnostic Tools and ECUs

In the automotive industry, there are numerous vehicle manufacturers and ECU suppliers. Each manufacturer may use different tools for diagnostics, leading to compatibility issues. UDS provides a standardized framework to ensure that diagnostic tools can communicate with any ECU that implements UDS, but implementing this standard requires attention to detail to ensure full compliance and interoperability across all systems.

The primary problem this project aims to solve is the lack of a standardized, embedded diagnostic solution that can efficiently communicate with multiple ECUs in a vehicle using UDS over the CAN bus. By implementing UDS on an STM32 microcontroller, the project will provide a prototype solution that demonstrates how standardized diagnostic communication can improve vehicle fault detection, system monitoring, and ECU reprogramming, all while ensuring realtime performance and compliance with automotive industry standards.

PROPOSED SYSTEM

The project involves the design and implementation of the Unified Diagnostic Services (UDS) protocol on an embedded platform, specifically targeting vehicle diagnostics over the Controller Area Network (CAN) bus. The proposed methodology will focus on the development of a UDS software stack, integrating it with the hardware (STM32 microcontroller) and ensuring communication with external diagnostic tools via CAN. The proposed method follows the next-mentioned steps.

4.1. OSI model for UDS

The Unified Diagnostic Services (UDS) protocol follows the ISO 14229 standard, and its implementation can be mapped into the Open Systems Interconnection (OSI) model. This mapping shows how UDS interacts with various layers, particularly for communication over the Controller Area Network (CAN).

OSI Model Layers	UDS Protocol
Application	UDSonCAN ISO 142291 ISO 142293
Presentation	NA
Session	UDS ISO 142292
Transport	ISOTP ISO 157652
Network	ISOTP ISO 157652
Data link	CAN ISO 118982
Physical	CAN ISO 118982

Fig.1. UDS protocol implementation as per OSI model

4.1.1. Application Layer (Layer 7)

UDS operates here, providing diagnostic services like Read Data by Identifier (0x22) and Diagnostic Session Control (0x10).

4.1.2. Session Layer (Layer 5)

Manages diagnostic sessions (e.g., default, extended, programming). UDS switches between these sessions to control access.

4.1.3. Transport Layer (Layer 4)

Uses ISOTP (ISO 157652) to handle message fragmentation and flow control over CAN.

4.1.4. Network Layer (Layer 3)

ISOTP manages logical addressing and reassembly of CAN frames, even without traditional IP addressing.

4.1.5. Data Link Layer (Layer 2)

The CAN protocol (ISO 11898) frames UDS messages, handles error checking (CRC), and bus arbitration.

4.1.6. Physical Layer (Layer 1)

Transmits CAN signals via CAN bus wiring and CAN transceivers.

This mapping ensures UDS runs efficiently over CAN with reliable communication between diagnostic tools and ECUs.

4.2. Hardware Selection and Setup

The primary hardware platform will be the STM32 microcontroller, which will handle the UDS communication and CAN protocol interface. The CAN transceiver will be used to facilitate communication between the microcontroller and the vehicle's CAN network or a simulated CAN bus environment. A PC-based diagnostic tool will be used to send diagnostic requests and receive responses from the STM32 microcontroller.

4.2.1 Hardware Components:

- STM32 Microcontroller: For running the UDS protocol stack.
- CAN Transceiver: For CAN communication.
- PC-based Diagnostic Tool: For sending diagnostic commands and receiving responses.

4.3. Software Design: UDS Stack Implementation

The core of the project is the implementation of the UDS protocol stack on the STM32 microcontroller. This stack will include the following components:

4.3.1 Transport Layer (ISOTP)

Implements message fragmentation and reassembly for UDS over the CAN bus.

4.3.2 UDS Services

Implementation of key UDS services such as reading fault codes, ECU reset, and reprogramming.

4.3.3.CAN Interface

Software drivers for the CAN controller to send and receive CAN frames.

The UDS stack will be designed in compliance with ISO 14229, handling requests from the diagnostic tool and sending appropriate responses.

4.4. UDS Message Structure

UDS is a request-based protocol.

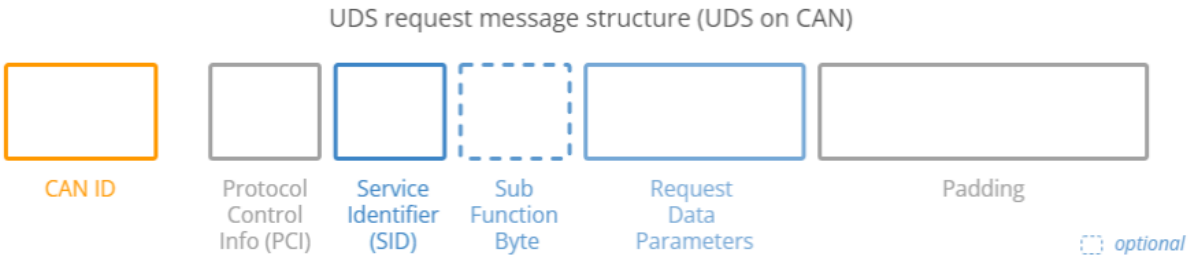

Fig.2. UDS Message Structure

4.4.1. Protocol Control Information (PCI)

The PCI field is not per se related to the UDS request itself but is required for diagnostic UDS requests made on CAN bus. In short, the PCI field can be 13 bytes long and contains information related to the transmission of messages that do not fit within a single CAN frame.

4.4.2. UDS Service ID (SID)

The use cases outlined above relate to different UDS services. When you wish to utilize a specific UDS service, the UDS request message should contain the UDS Service Identifier (SID) in the data payload. Note that the identifiers are split between request SIDs (e.g. 0x22) and response SIDs (e.g. 0x62).

4.4.3. UDS Sub Function Byte

The sub-function byte is used in some UDS request frames as outlined below. Note, however, that in some UDS services, like 0x22, the sub-function byte is not used. Generally, when a request is sent to an ECU, the ECU may respond positively or negatively.

4.4.4. UDS 'Request Data Parameters' incl. Data Identifier (DID)

In most UDS request services, various types of request data parameters are used to provide further configuration of a request beyond the SID and optional sub-function byte.

4.5. Positive vs. negative UDS responses

When an ECU responds positively to a UDS request, the response frame is structured with similar elements as the request frame. For example, a 'positive' response to a service 0x22 request will contain the response SID 0x62 (0x22 + 0x40) and the 2byte DID, followed by the actual data payload for the requested DID. Generally, the structure of a positive UDS response message depends on the service. However, in some cases an ECU may provide a negative response to an UDS request for example if the service is not supported.

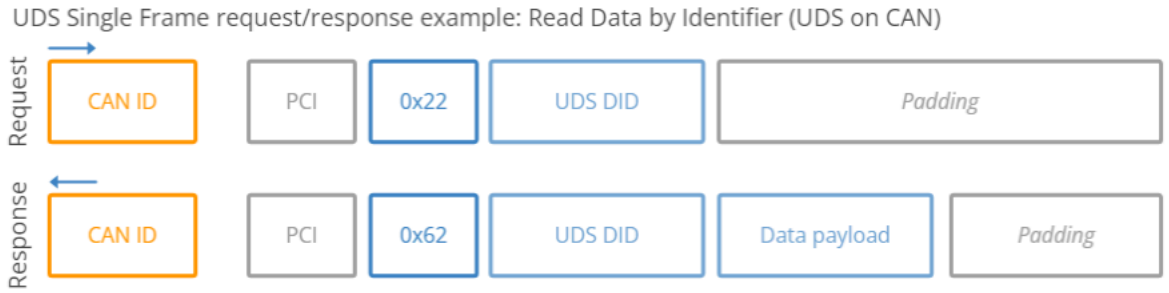

Fig.3. UDS Request and Response message structure

4.6. Client Server Architecture in UDS

In the UDS protocol, communication follows a client-server architecture. For the proposed system, this architecture enables the exchange of diagnostic information between the diagnostic tool (client) and the vehicle's ECU (server) over the CAN bus.

4.6.1 Client (Diagnostic Tool)

The client initiates diagnostic requests. It sends commands to the ECU for actions like reading data, clearing diagnostic trouble codes (DTCs), or performing ECU reprogramming. The client is typically a diagnostic scanner or testing equipment.

4.6.2 Server (ECU)

The server, which is the Electronic Control Unit (ECU), processes the client's requests and responds accordingly. It provides data or executes actions as specified by the client's UDS commands, such as responding with vehicle status or resetting the ECU.

This architecture ensures seamless communication between diagnostic tools and the vehicle's systems for diagnostic and maintenance operations.

4.7. UDS Service Implementation

- Diagnostic Session Control (0x10): Allows switching between different diagnostic modes.
- Read Data by Identifier (0x22): Retrieves specific vehicle data.
- Clear Diagnostic Information (0x14): Clears diagnostic trouble codes (DTCs) stored in the ECUs.
- Request Download (0x34) and Transfer Data (0x36): Facilitates ECU programming or software updates.

These UDS services will be implemented in the microcontroller, allowing interaction between the vehicle and diagnostic tools.

4.8. Testing and Simulation

Once the UDS protocol stack is implemented, the system will be tested in a controlled environment. A CAN bus simulator or a real vehicle CAN network will be used to verify the functionality of the UDS services. The PC based diagnostic tool will send UDS requests, and the microcontroller will process the requests and respond according to the UDS protocol.

4.8.1 Testing Focus

- Verification of UDS services over the CAN network.
- Realtime performance of fault detection and diagnostic requests.
- ECU reprogramming tests using UDS services.

BLOCK DIAGRAM

The block diagram below illustrates the overall architecture of the proposed method for implementing the UDS protocol over CAN using the STM32 microcontroller.

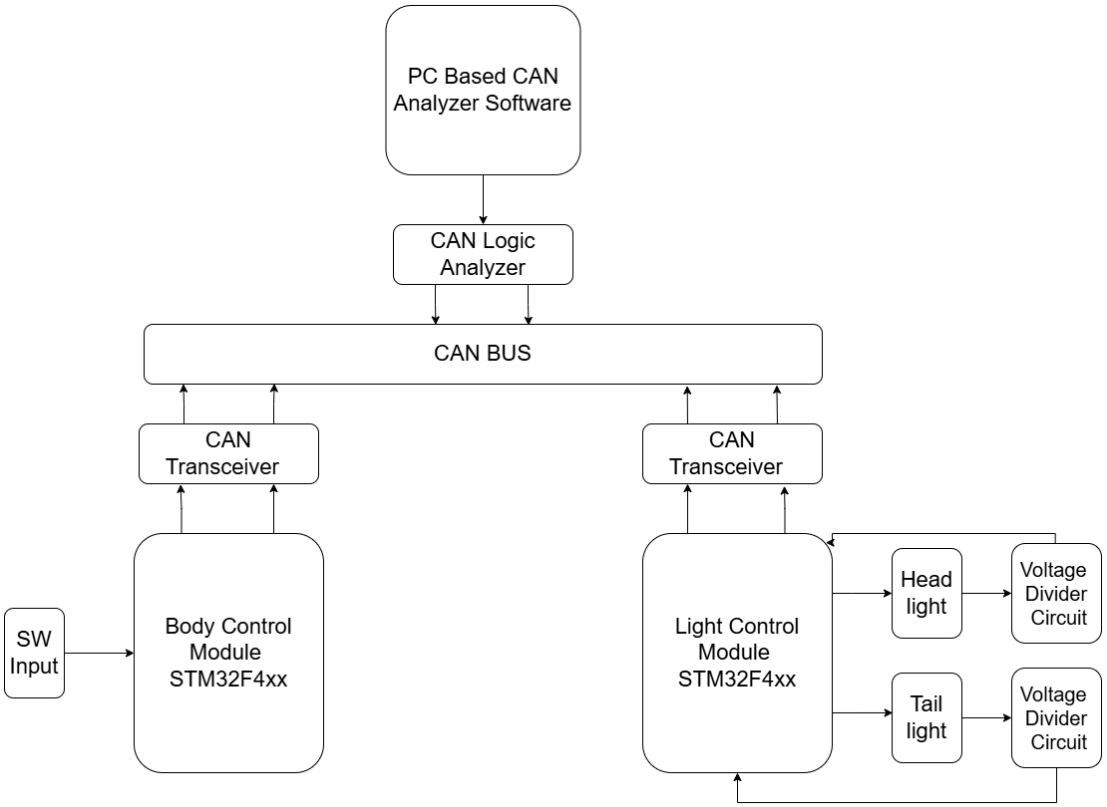

Fig. 4. Block diagram of UDS protocol implementation

5.1. PCBased Diagnostic Tool (UDS Client)

This tool sends diagnostic requests (e.g., reading fault codes, clearing errors) to the vehicle's ECUs via the CAN bus. It acts as the UDS client, initiating communication with the embedded system.

5.2. CAN Transceiver (MCP2515)

The transceiver serves as the communication bridge between the PCbased diagnostic tool and the STM32 microcontroller, converting the CAN signals to a format the microcontroller can process.

5.3 STM32 Microcontroller (UDS Server)

This is where the core UDS protocol stack is implemented. The microcontroller processes UDS requests, executes services (like reading data or ECU reprogramming),

and sends responses back to the diagnostic tool. It handles the ISOTP layer to manage larger CAN messages and ensure proper data exchange.

5.4. Switch Input

The input for the Body control module is from switches. Four leg push buttons are used as input for the BCM module.

5.5. Headlight and Taillight

The headlight and Taillight are the output of the system. They are connected to the Light control module (LCM). To check whether the led for headlight and taillight are functional or not voltage divider circuit is used.

FLOWCHART

6.1 Read Data by Identifier (0x22)

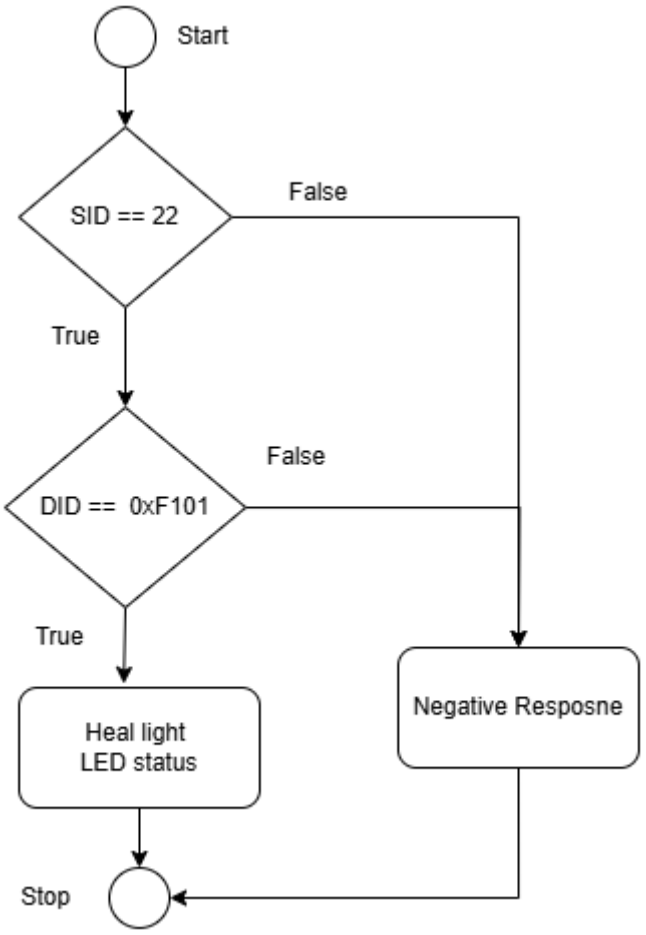

Fig.5. Flowchart for Read Data by Identifier

In the flow chart for UDS Service 0x22 (Read Data by Identifier), the process begins with the ECU receiving a request message that includes Service ID 0x22 and a DID (Data Identifier) to read. The ECU then checks if the requested DID is supported. If the DID is valid, the ECU retrieves the corresponding data from memory or storage associated with that DID and sends a positive response with Service ID 0x62 followed by the DID and its value. If the DID is unsupported or unavailable, the ECU responds with a negative response (0x7F) and an appropriate error code. The flow then returns to an idle or wait state until a new request is received.

6.2 Write Data by Identifier (0x2E)

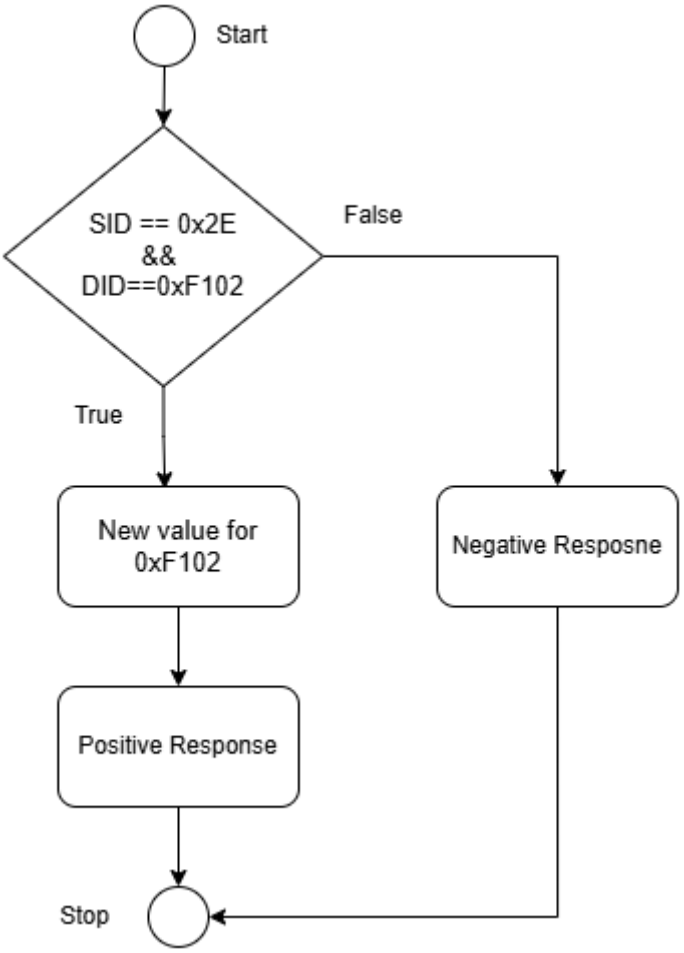

Fig.6. Flowchart for Write Data by Identifier

The flow chart for the UDS Service 0x2E (Write Data by Identifier) begins with the ECU receiving a request containing the DID (Data Identifier) and data to write. The ECU first validates the DID to check if it's supported. If the DID is valid, it proceeds to write the data to the specified memory location or register. After writing, the ECU sends a positive response with a response code (0x6E or 0x62), indicating success. If the DID is invalid or the data cannot be written, the ECU sends a negative response (0x7F) with an error code, such as "Request Out of Range" (0x31) or "Sub-function Not Supported" (0x12), before ending the process. This flow ensures data integrity by verifying the DID and handling errors gracefully.

6.3 Routine Control (0x31)

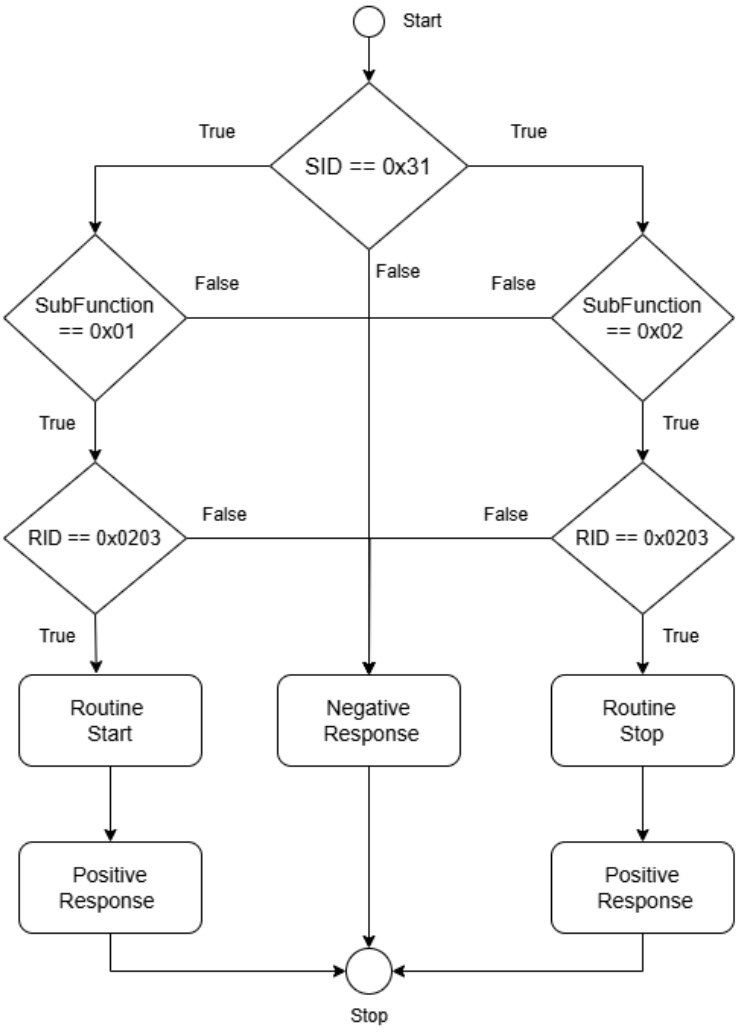

Fig.7. Flowchart for Routine Control

In the flow chart for UDS Service 0x31 (Routine Control) with Start (0x01) and Stop (0x02) subfunctions, the process begins with the ECU receiving a UDS request containing the service ID (0x31), subfunction, and Routine Identifier (RID). First, the ECU extracts and validates the RID. If the RID matches the target routine (e.g., 0x0100), the ECU then checks the subfunction. For Start (0x01), the ECU initiates the routine (e.g., turning on an LED or starting a process) and sends a positive response with response code 0x71. For Stop (0x02), it halts the routine (e.g., turning off the LED or stopping the process) and sends response code 0x72. If the RID is invalid or the subfunction is unsupported, the ECU sends a negative response (0x7F) with an error code, ending the process.

Application

The implementation of the Unified Diagnostic Services (UDS) protocol in automotive systems opens up several important applications in the automotive industry, providing benefits to manufacturers, service centers, and vehicle owners. Some key applications include

7.1. Vehicle Diagnostics and Troubleshooting

UDS is primarily used for diagnosing issues within a vehicle's electronic control units (ECUs). Technicians can connect diagnostic tools to the vehicle's onboard systems, allowing them to:

- Retrieve diagnostic trouble codes (DTCs) that indicate malfunctions or component failures.
- Perform realtime monitoring of sensor data and ECU parameters.
- Access historical fault records for accurate diagnosis.
- Test specific functions (e.g., engine misfire detection) without physically dismantling the vehicle.

7.2. ECU Reprogramming and Software Updates

UDS facilitates secure reprogramming and flashing of ECUs. With this functionality, automotive manufacturers can update the vehicle's software to fix bugs, add new features, or improve performance. UDS also supports:

- Over The Air (OTA) updates, reducing the need for service visits.
- Flashing of new firmware to rectify software related faults.
- Parameterization of ECUs to optimize vehicle performance or update configurations for specific markets.

7.3. Emissions Testing and Compliance

UDS is essential for accessing emissions related data to ensure that vehicles comply with environmental regulations such as OBD II in the United States and the European Union's EOBD standards. It allows:

- Reading of emission related trouble codes and realtime emissions data.
- Ensuring that vehicles meet regulatory emissions limits.
- Clearing trouble codes related to emissions systems after repairs.

7.4. InVehicle Networking and Communication Control

UDS supports controlling the communication between ECUs in the vehicle. This enables the diagnostic tool to manage and modify communication parameters such as baud rate, request and response cycles, and timeouts. The protocol can:

- Adjust communication settings to optimize data exchange.
- Manage multiple communication sessions between diagnostic tools and ECUs, ensuring safe and controlled access to critical data.

7.5. Security and Access Control

UDS provides robust security features to prevent unauthorized access to sensitive ECU functions. This includes:

- Managing access levels for different diagnostic operations, such as regular maintenance versus ECU reprogramming.
- Protecting against cyber threats by ensuring that only authorized personnel can access critical functions, such as reflashing firmware.
- Secure communication for remote diagnostics or OTA updates.

The implementation of the UDS protocol is crucial for modern vehicle diagnostics, ECU programming, and overall vehicle health management. It offers a standardized and secure method for accessing a wide range of diagnostic services, from simple fault detection to advanced ECU reprogramming and security access. This project will provide key advantages to the automotive industry, helping streamline maintenance, reduce downtime, and ensure compliance with industry regulations.

Expected Results

The project aims to implement a fully functional Unified Diagnostic Services (UDS) protocol over the CAN bus on an STM32 microcontroller. The expected results include not only the correct functioning of the UDS protocol stack but also the successful execution of UDS services with proper request and response handling. Below are the key services that will be implemented, along with their corresponding request and response formats.

8.1 DID for Headlight Status Check

Identifier: 0xF101

Purpose Check whether the headlight is ON or OFF and retrieve status information.

Example Use Case

Request: The tester requests the headlight status by sending a 0x22 service (Read Data By Identifier) followed by 0xF101.

Request Frame: 0x22 0xF1 0x01

Response: The ECU responds with the status (e.g., 0x01 for ON, 0x00 for OFF).

Response Frame: 0x62 0xF1 0x01 0x01 (Headlight ON)

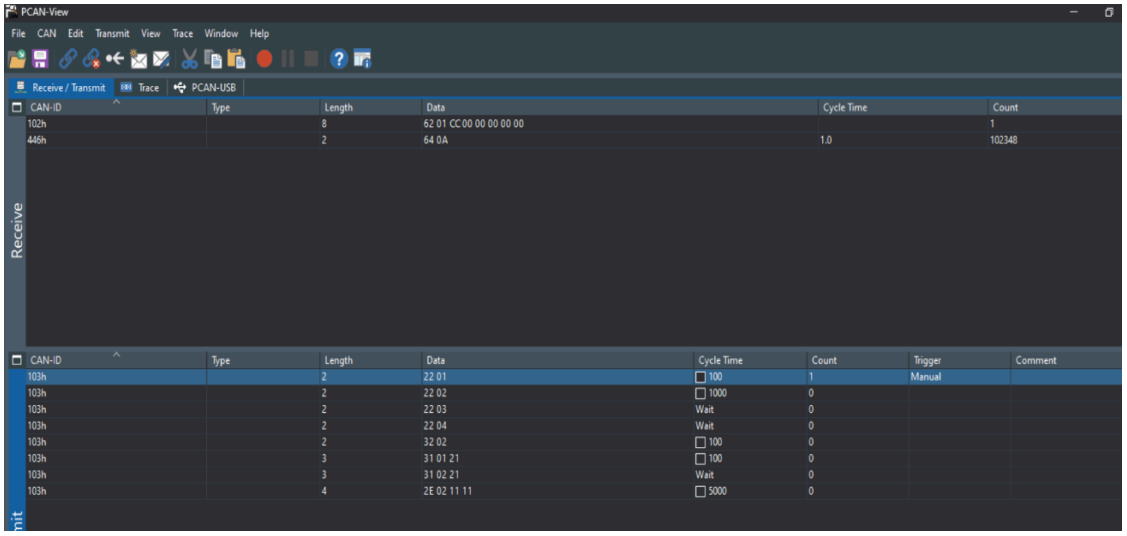

Fig.8. Expected result for Read Data by Identifier

8.2 DID for Blinking Delay Adjustment

Identifier: 0xF102
Purpose: Adjust the delay time between LED blinks.

Example Use Case

Request: The tester sends a 0x2E service (Write Data By Identifier) with 0xF102 and specifies a time delay value (e.g., 0x05 for a 500ms delay).

Request Frame: 0x2E 0xF1 0x02 0x05

Response: The ECU confirms by echoing the DID and the new value.

Response Frame: 0x6E 0xF1 0x02 0x05 (500ms delay set)

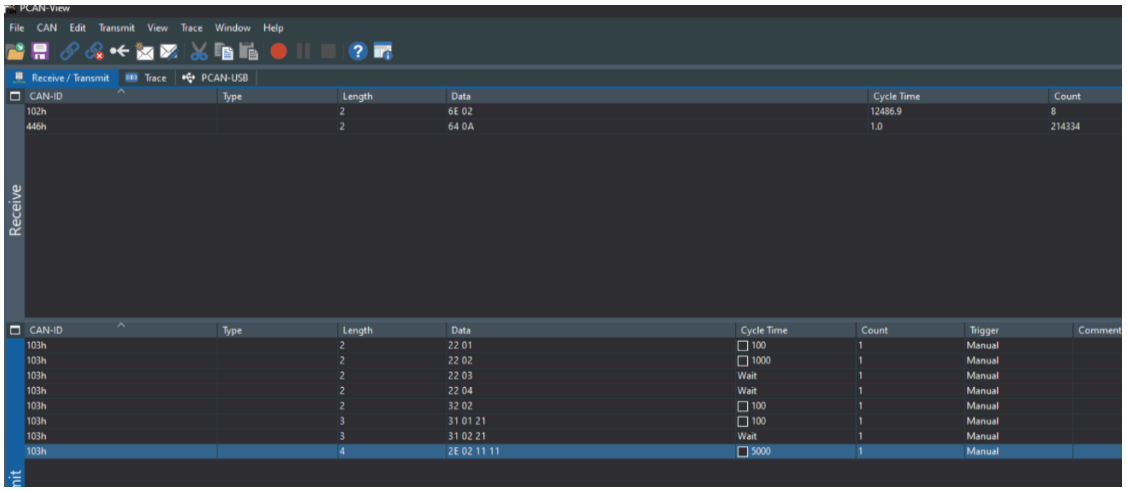

Fig.9. Expected Result of Write Data by Identifier

8.3 RID for Predefined Lighting Sequence

Identifier: 0x0203

Purpose: Trigger a diagnostic lighting sequence to verify the functionality of the lighting control.

Example Use Case

Request: The tester sends a 0x31 service (Routine Control) with 0x0203 and the start subfunction 0x01 to initiate the routine.

Request Frame: 0x31 0x01 0x02 0x03

Response: The ECU acknowledges the routine start and runs the lighting sequence.

Response Frame: 0x71 0x01 0x02 0x03

Stop Routine: To end the sequence, the tester sends 0x31 with a stop subfunction 0x02.

Stop Frame: 0x31 0x02 0x02 0x03

Response: 0x71 0x02 0x02 0x03

8.4 DTC for CAN Communication Error

DTC Code: 0xU1000

Description: Indicates a communication issue between ECUs over the CAN bus.

Example Use Case

Detection: If the CAN bus connection is interrupted, the BCM detects a loss of communication with the LCM.

Read DTC Request: The tester requests the DTC information with 0x19 0x02.

Request Frame: 0x19 0x02

Response: The ECU responds with the DTC 0xU1000.

Response Frame: 0x59 0x02 0xU1 0x00 0x00 (indicating CAN Communication Error)

8.5 DTC for Headlight LED Fault

DTC Code: 0xB105

Description: Indicates an open circuit or fault in the headlight LED.

Example Use Case

Detection: If the headlight LED is non-functional, the LCM registers 0xB105 as a DTC.

Read DTC Request: The tester sends a 0x19 0x02 request to retrieve DTCs.

Request Frame: 0x19 0x02

Response: The ECU responds with the DTC 0xB105.

Response Frame: 0x59 0x02 0xB1 0x05

8.6 DTC for ECU Power Off Condition

DTC Code: 0xB20A

Description: Indicates that the LCM has powered off unexpectedly or is unresponsive.

Example Use Case

Detection: The BCM logs a 0xB20A code when it fails to receive responses from the LCM for a predefined period.

Read DTC Request: Tester queries DTCs with 0x19 0x02.

Request Frame: 0x19 0x02

Response: The ECU responds with the DTC 0xB20A.

Response Frame: 0x59 0x02 0xB2 0x0A

CONCLUSION

In conclusion, this Design and Implementation of UDS Protocol for Automotive Diagnostic project successfully implemented diagnostic communication on an STM32F407 controller to interface with a Body Control Module (BCM) and a Lighting Control Module (LCM) over CAN Bus. Key UDS services, including Read Data by Identifier (0x22), Write Data by Identifier (0x2E), and Routine Control (0x31), were developed to manage and monitor BCM and LCM functionalities, such as reading status data, configuring parameters, and controlling routines like turning headlights on or off. The CAN Bus enabled reliable communication between ECUs, simulating realistic automotive diagnostic scenarios. Overall, this project provides a foundational UDS setup for testing, monitoring, and troubleshooting BCM and LCM systems within a CAN network, adhering to industry diagnostic standards.

REFERENCES

[1] U. Kataria, K. Panchal, J. Puvvada, S. Kadlag and S. Shinde, “ Implement Standard UDS Diagnostics Over Can for Automotive Actuator ECU”, 2024 International Journal for Multidisciplinary Research (IJFMR), Mumbai, India, E-ISSN: 2582-2160, IJFMR240217526.

[2] M. Kuntoji, V. Medam and Veena Devi S. V, “ Design of UDS Protocol in an Automotive Electronic Control Unit”, 2023 Recent Developments in Electronics and Communication Systems, Bangalore, India, doi:10.3233/ATDE221266

[3] S. Desai and Y. Bhateshvar, “ Development of unified diagnostic services on CAN using MATLAB and Arduino”, 2023 Materials Today: Proceedings 72 (2023) 1935–1942, Pune, India, https://doi.org/10.1016/j.matpr.2022.10.157

[4] ISO-14229: International Standard - Road vehicles - Unified diagnostic services (UDS), 2006.

[5] ISO 11898-1:2024 - Road vehicles – Controller area network (CAN) — Part 1: Data link layer and physical coding sublayer.

[6] ISO 15765-2:2016 Road vehicles – Diagnostic communication over Controller Area Network (DoCAN) -- Part 2: Transport protocol and network layer services.

[7] ISO 13400-1:2011-Road vehicles – Diagnostic communication over Internet Protocol (DoIP): — Part 1: General information and use case definition. — Part 2: Transport protocol and network layer services.

[8] ISO 26262-1:2011- Road vehicles – Functional Safety

[9] R. Chatterjee, C. Green and J. Daily, “Exploiting Diagnostic Protocol Vulnerabilities on Embedded Networks in Commercial Vehicles”, 2024, Symposium on Vehicles Security and Privacy (VehicleSec) 2024, USA, ISBN 979-8-9894372-7-6.

[10] V. N. D. Krishnamurthy, “ Unified Diagnostic Services (UDS) in Automotive: A technical Study”, 2021, European Journal of Advances in Engineering and Technology, 2021, 8(6):70-73, USA, ISSN: 2394-658X