INTERNATIONAL SCIENTIFIC JOURNAL OF ENGINEERING AND MANAGEMENT (ISJEM) ISSN: 2583-6129
VOLUME: 04 ISSUE: 05 | MAY – 2025 DOI: 10.55041/ISJEM03371
AN INTERNATIONAL SCHOLARLY || MULTIDISCIPLINARY || OPEN ACCESS || INDEXING IN ALL MAJOR DATABASE & METADATA

Design and Development of Automotive On-Board Diagnostic Protocol system

Priti Dattaray Hagawane
Department of Electronics and Telecommunication
SVPM College of Engineering, Baramati, Pune
hagawanepriti@gmail.com

Dr. Nitin B. Dhaygude
Department of Electronics and Telecommunication
SVPM College of Engineering, Baramati, Pune
nbdhaigude@engg.svpm.org.in

***Abstract*- The increasing complexity of modern vehicles necessitates the use of multiple Electronics Control Units (ECUs) to manage and monitor various functions. Communication between these ECUs is facilitated by the Controller Area Network (CAN) protocol, which handles the Physical and Data Link layers of the OSI model but lacks advanced diagnostic capabilities. To address this, standardized diagnostic protocols such as On-Board Diagnostics (OBD2) have been developed, providing a comprehensive framework for fault detection and in-vehicle communication. This paper presents the design and implementation of the OBD2 protocol on an embedded system using STM32F407 microcontrollers. The project aims to establish a diagnostic communication system between ECUs and Tester Tool. Primary function of this ECU is to measure ambient temperature and Throttle Position. ECU and Tester tools are connected via a CAN bus. The implementation leverages the ISO 15031 standard to ensure reliable and standardized diagnostic services.**

Keywords: On-Board Diagnostics (OBD2), ISO 15031, Electronic Control Unit (ECU), Body Control Module (BCM), Light Control Module (LCM), Controller Area Network (CAN), Diagnostic Trouble Code (DTC), Routine Control Identifier (RID), Data Identifier (DID), ISO-TP (Transport Protocol)

I. INTRODUCTION

The complexity of modern vehicles necessitates advanced diagnostic capabilities to ensure optimal performance and reliability. The OBD2 protocol, standardized under ISO 15031, is a crucial tool in the automotive industry for monitoring and diagnosing vehicle systems. OBD2 provides a comprehensive framework for fault detection, emission control, and real-time data access, facilitating efficient vehicle maintenance and repair.

This paper explores the design and implementation of an OBD2-based diagnostic communication system using STM32F407 microcontrollers. The project involves two key components one is ECU and another is PC Based tester tool. ECU and Tester tools communicate over a CAN bus and respond to OBD2 diagnostic requests. The system supports vital OBD2 services, including retrieving DTCs(0x03) and accessing real-time sensor data. Mode (0x01)

To ensure the system's robustness, fault injection techniques are employed to simulate real-world ECU failures such as over Temperature and ECU power loss. The diagnostic responses to these failures are analyzed using a Waveshare USB-to-CAN module. By implementing OBD2 on embedded automotive ECUs, this project aims to demonstrate the practical application of standardized diagnostic services, enhancing the efficiency of vehicle fault detection and maintenance processes.

II. LITERATURE SURVEY

Malekian et al. designed a wireless OBD-II fleet management system using a custom OBD reader to measure real-time vehicle data such as speed and fuel consumption, and then transmitted the data to a remote server via Wi-Fi [1]. Their work highlights the ability of OBD-II to provide accurate data using PID requests, although the focus was more on wireless transmission and cloud integration.

Baek and Jang implemented an integrated OBD-II connector that supports Bluetooth, Wi-Fi, and WCDMA modules for multi-platform communication [2]. While this study focused on external interfacing, the underlying diagnostic protocol used SAE J1979 and ISO 15765-4 standards for CAN-based request-response communication with the ECU.

Joseph and Kumar presented a low-cost Driver Information System (DIS) that reads live OBD-II data such as throttle position, RPM, and ambient temperature using standard PID requests [3]. Their system used a polling mechanism to repeatedly query the ECU, decode the hexadecimal responses, and present the results in a user-friendly format, demonstrating a structure similar to the STM32-based system in this project.

Jhou et al. implemented a cloud-based OBD-II diagnostic system using a 3.5G wireless module to send vehicle data to a cloud server for real-time fault classification [4]. Although the project emphasized cloud computation, the use of standard services like Mode 01 (Current Data) and Mode 03 (DTCs) forms a strong parallel to the core services implemented in the present project.

Moniaga et al. conducted a study on real-time vehicle diagnostics using an OBD-II scanner with a Raspberry Pi platform [5]. The system acquired data such as throttle position and RPM and displayed it on a local interface. The comparison between microcontrollers (Arduino vs. Raspberry Pi) underscored the importance of processing capability in real-time OBD-II applications.

Michael and Sunday conducted a case study on digitized vehicle diagnostics using tools like Launch X431 and Autel MaxiDiag to capture and interpret fault codes and vehicle parameters [6]. Their findings confirm that OBD-II is effective in identifying malfunction conditions early, which reduces emissions and improves vehicle maintenance.

OBD-II is a standardized vehicle diagnostics protocol that provides access to real-time vehicle data and DTCs via a 16-pin standardized connector. The core functionality is defined in SAE J1979 / ISO 15031-5 for diagnostic services and PIDs, ISO 15765-4 for CAN-based communication, and ISO 15031-3 / SAE J1962 for connector specifications. These standards ensure uniformity across all OBD-II-compliant vehicles and provide the technical basis for diagnostic tools and embedded implementations.. ISO 15765-4 defines how CAN is used for diagnostics, while ISO 15765-2 defines the transport protocol for multi-frame messaging. ISO 15031-5 (SAE J1979) lists the standard services and PIDs, and ISO 15031-3 (SAE J1962) defines the standardized connector, ensuring tool compatibility across manufacturers [7]–[10].

III. SYSTEM OVERVIEW

A. OBD2 Protocol Overview

OBD2 is a standardized system implemented in modern vehicles to monitor and diagnose various aspects of vehicle performance and emissions. Defined under the ISO 15031 standard, OBD2 provides a comprehensive framework for fault detection, real-time data access, and emission control, ensuring vehicles meet regulatory requirements and maintain optimal performance. The OBD2 protocol enables communication between the vehicle's ECUs and diagnostic tools via the CAN bus.

B. Client-Server Architecture

The OBD2 protocol operates on a client-server architecture, where the diagnostic tool acts as the client and the vehicle's ECUs act as servers. Client initiates diagnostic request, and Server will respond to the request as per configuration. The diagnostic tool provides a user-friendly interface for mechanics and technicians to interact with the vehicle's diagnostic system. This interface allows users to select specific tests, view data, and perform diagnostic procedures. Client-Server architecture provides standardization, Efficiency, Scalability and Flexibility.

C. OBD2 Message Structure

an OBD2 message consists of an identifier and data. Further, the data is split in Mode, PID and data bytes.

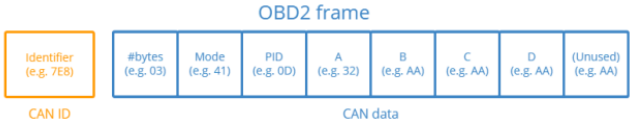


Fig.1. OBD2 Protocol Frame Structure

- Identifier: For OBD2 messages, the identifier is standard 11 bit and used to distinguish between "request messages" (ID 7DF) and "response messages" (ID 7E8 to 7EF). Note that 7E8 will typically be where the main engine or ECU responds at.
- Length: This simply reflects the length in number of bytes of the remaining data (03 to 06). For the Vehicle Speed example, it is 02 for the request (since only 01 and 0D follow), while for the response it is 03 as both 41, 0D and 32 follow.
- Mode: For requests, this will be between 010A. For responses the 0 is replaced by 4 (i.e. 41, 42, … , 4A). There are 10 modes as described in the SAE J1979 OBD2 standard. Mode 1 shows Current Data and is e.g. used for looking at real-time vehicle speed, RPM etc. Other modes are used to e.g. show or clear stored DTC and show freeze frame data.
- PID: For each mode, a list of standard OBD2 PIDs exist e.g. in Mode 01, PID 0D is Vehicle Speed. For the full list, check out our OBD2 PID overview . Each PID has a description and some have a specified min/max and conversion formula. The formula for speed is e.g. simply A, meaning that the A data byte (which is in HEX) is converted to decimal to get the km/h converted value (i.e. 32 becomes 50 km/h above). For e.g. RPM (PID 0C), the formula is (256A + B) / 4.
- A, B, C, D: These are the data bytes in HEX, which need to be converted to decimal form before they are used in the PID formula calculations. Note that the last data byte (after Dh) is not used.

D. OBD2 Request

An OBDII request is a message sent by a diagnostic tool or scan tool to the vehicle's ECU to request specific diagnostic information. The request typically consists of several parts. Mode Defines the type of operation or request being made. OBDII defines several modes, each with a specific purpose. For example

1. Mode 01: Request Current Data
2. Mode 02: Request Freeze Frame Data
3. Mode 03: Request DTC
4. Mode 04: Clear DTC
5. Mode 05: Request Oxygen Sensor Monitoring
6. Mode 06: Request Test Results for Specific Monitors
7. Mode 07: Request Pending DTCs
8. Mode 08: Request Control of OnBoard System
9. Mode 09: Request Vehicle Information

PID (Parameter ID) Specifies the particular data or diagnostic information requested within the given mode. For example, in Mode 01 (Request Current Data), PIDs represent specific parameters like engine RPM, vehicle speed, coolant temperature, etc. Data Bytes In some cases, additional data bytes are sent with the request to provide parameters or specific instructions.

Example Request To request the current engine RPM, a diagnostic tool would send a request with Mode 01 and PID 0C (which represents RPM data).

E. OBD2 Response

An OBDII response is the message sent by the vehicle's ECU back to the diagnostic tool in reply to a request. The response contains the requested diagnostic data or an indication of the operation's success or failure. The response typically consists of

- Mode Repeats the mode of the request to confirm the type of data being returned.
- PID Identifies the specific parameter or diagnostic information returned.

- Data Bytes Contains the actual diagnostic data requested. The format and content of these bytes depend on the PID and the requested data.
- Status Byte Sometimes included to indicate the status of the request, such as whether it was successful or if there were errors.

Example Response For a request of current engine RPM (Mode 01, PID 0C), the response might include a data byte sequence representing the RPM value. For instance, the response data might be 0C 0A 00 1E, where 0A is a status byte and 001E is the RPM value (in hexadecimal format).

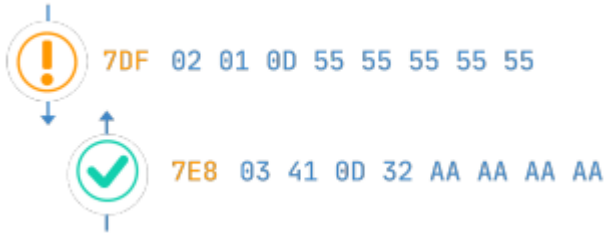


Fig.2. OBD II Request and Response message structure

F. *Block Diagram*

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

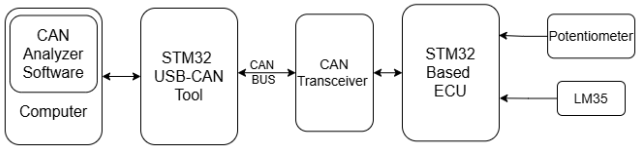


Fig. 3. Block diagram of OBD protocol implemented system

- Sensors (Potentiometer and LM35): A potentiometer is connected to the STM32-based ECU to simulate the throttle position sensor, providing variable analog input based on its rotation. An LM35 temperature sensor is used to measure the ambient temperature, supplying a linear analog voltage corresponding to the temperature in degrees Celsius.
- STM32-Based ECU: The STM32F407VG microcontroller acts as the ECU, responsible for reading analog signals from the sensors through its built-in ADC. It processes these values and converts them into standard diagnostic data, formatted according to the OBD2 protocol. In terms of OBD2 protocol, the STM32 based controller is called a server, which processes the OBD2 Request and provides a response.
- CAN Transceiver(MCP2551) : This module enables communication between the STM32 microcontroller and the CAN bus.It converts the controller's TX/RX logic levels to CAN differential signals and vice versa.The processed data is transmitted over the CAN bus via a CAN transceiver, which converts the digital signals from the STM32 into differential signals suitable for CAN communication and vice versa.
- STM32 USB-CAN Tool (Waveshare): This acts as a bridge between the CAN bus and the PC. It captures the CAN frames sent by the STM32 ECU and transmits them to the computer via USB. The CAN transceiver is connected to an STM32-based USB-to-CAN tool, which acts as a bridge between the CAN bus and the computer. It enables communication between the ECU and diagnostic software on the PC.
- Computer with CAN Analyzer Software: The PC runs CAN analyzer software to send diagnostic requests (OBD2 service requests) and receive the corresponding responses from the ECU. It helps in visualizing and verifying the communication between the diagnostic tool (tester) and the ECU.

IV. FLOW CHART

The flowcharts represent the communication flow for OBD-II Services 01 and 03 implemented on an STM32 microcontroller. The process includes CAN interface initialization, diagnostic request generation, message transmission, and ECU response handling. Service 01 is used to access real-time vehicle parameters, while Service 03 retrieves stored DTCs. Each flow ensures structured data exchange over the CAN protocol, enabling accurate interpretation and display of diagnostic data.

A. *Mode 01 Request Current Data*

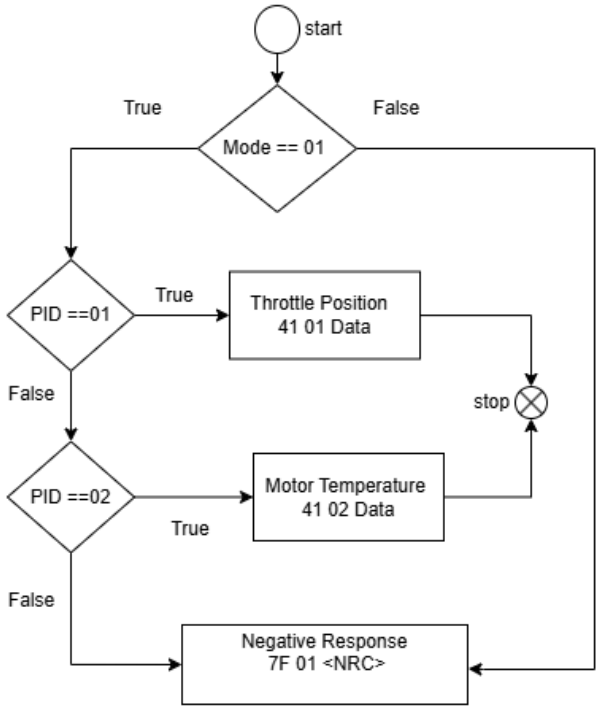


Fig. 4. Flowchart for OBD Mode 01 Request current Data

In OBD Mode 01 (Request Current Data), the flow begins with the scan tool (diagnostic tester) sending a Mode 01 request with a PID (Parameter ID), which specifies the type of real-time data it needs from the vehicle's ECU, such as engine speed, vehicle speed, or coolant temperature. Upon receiving the request, the ECU interprets the PID, retrieves the relevant sensor data, and formats the response message.

The ECU then sends this response back to the scan tool, which decodes and displays the requested information for the user. This process enables real-time monitoring of vehicle parameters to assist in diagnostics and performance analysis.

B. Mode 03 Request Diagnostic Trouble Code

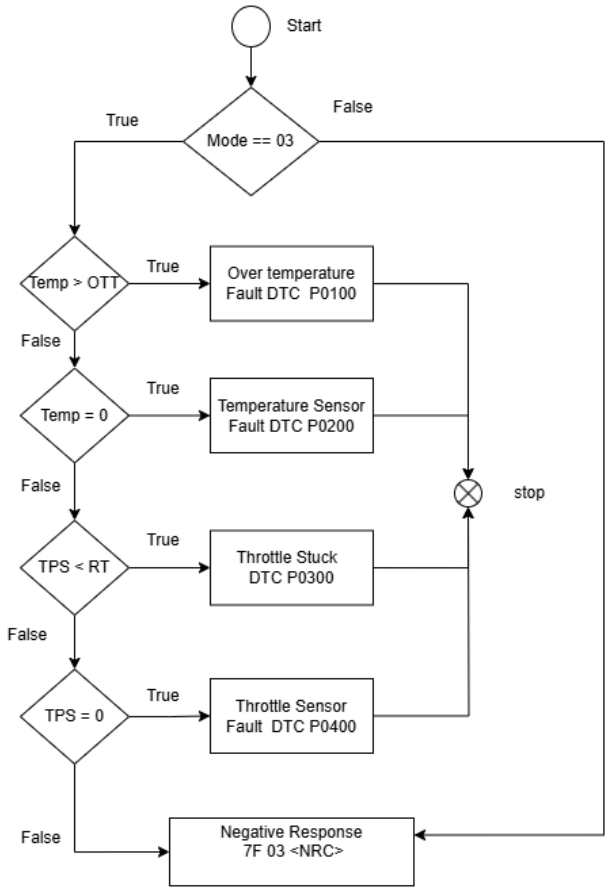


Fig.5. Flowchart for OBD Mode 03 Request DTC

The flow chart for OBD Mode 03 (Request DTCs) involves a sequence where the scan tool sends a request to the vehicle's ECU to retrieve DTC stored during fault detection. First, the scan tool initiates a request by sending Mode 03 over the OBD interface. The ECU receives this request, checks its internal memory for any stored DTCs, and formats the response accordingly. If DTCs are found, the ECU sends back a list of DTC codes, each identifying a specific fault. If no DTCs are present, the ECU responds with an indication of no faults. Finally, the scan tool displays the retrieved codes, allowing further diagnostics or repairs based on the fault information.

V. APPLICATION

The design and development of an Automotive On-Board Diagnostic (OBD) Protocol System has wide-ranging applications across vehicle diagnostics, maintenance, and performance optimization. It enables real-time data visualization and advanced analysis to improve diagnostic efficiency, helping technicians quickly identify and resolve issues through detailed insights into performance metrics and DTC. The system supports preventive maintenance by continuously monitoring key parameters to predict potential issues, and it also maintains comprehensive service histories for better long-term vehicle care. For performance optimization, the system provides real-time monitoring and historical data analysis to fine-tune vehicle settings. Additionally, it offers user-friendly interfaces for vehicle owners to track their car's health, enhancing overall vehicle management and reliability.

VI. RESULTS

The implementation of the OBD2 protocol on the STM32 microcontroller was tested using various diagnostic service requests sent via a PC-based CAN analyzer. The system was configured to simulate real ECU behavior, responding to Mode 01 (Show Current Data) and Mode 03 (Read Stored DTCs) requests.

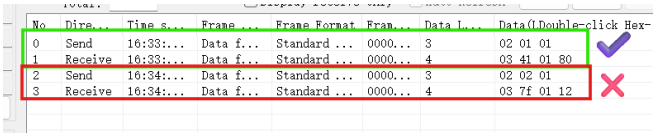
Sensor inputs were provided using an LM35 temperature sensor and a potentiometer, representing ambient temperature and throttle position, respectively. The ECU processed the analog signals and returned real-time values encoded into standard OBD2 PID response formats. Additionally, artificial faults were simulated by inducing specific sensor conditions such as over-temperature, disconnected sensors, or frozen input values. These faults were successfully logged as DTC (DTCs) and could be retrieved using Mode 03 requests.

The system also responded with negative responses when invalid or unsupported requests were received. According to the ISO 15031-5 / SAE J1979 standard, a negative response uses 0x7F as the service ID, followed by the requested service number and a Negative Response Code (NRC). This mechanism confirms that the ECU handles unexpected or incorrect requests gracefully, as per OBD2 protocol requirements.

A. Read Current sensor Data

The tester sends a request to read real-time data from the ECU using Mode 01 (Show Current Data).

- **PID 01**: Throttle Position (Potentiometer)

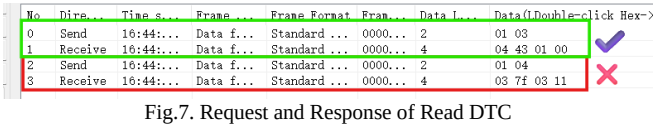


No	Dire...	Time s...	Frame ...	Frame Format	Fram...	Data L...	Data(Double-click Hex-
0	Send	16:33:...	Data f...	Standard ...	0000...	3	02 01 01
1	Receive	16:33:...	Data f...	Standard ...	0000...	4	03 41 01 80
2	Send	16:34:...	Data f...	Standard ...	0000...	3	02 02 01
3	Receive	16:34:...	Data f...	Standard ...	0000...	4	03 7f 01 12

Fig.6. Request and Response of Read Current Data

B. Request Diagnostic Trouble Code

This mode is used to read stored DTC that are related to emission system faults. It helps identify issues detected by the ECU that may cause increased emissions, aiding in quick fault diagnosis and repair.



No	Dire...	Time s...	Frame ...	Frame Format	Fram...	Data L...	Data(Double-click Hex->
0	Send	16:44:...	Data f...	Standard ...	0000...	2	01 03
1	Receive	16:44:...	Data f...	Standard ...	0000...	4	04 43 01 00
2	Send	16:44:...	Data f...	Standard ...	0000...	2	01 04
3	Receive	16:44:...	Data f...	Standard ...	0000...	4	03 7f 03 11

Fig.7. Request and Response of Read DTC

VII. CONCLUSION

The Design and Development of an Automotive On-Board Diagnostic (OBD) Protocol System successfully

demonstrates the ability to monitor, diagnose, and communicate vehicle faults, enhancing both the efficiency and reliability of automotive diagnostics. This project leveraged the standardized OBD protocol to facilitate real-time fault detection and reporting, allowing for quick identification of issues such as sensor malfunctions and system errors. Through structured implementation, the system achieved effective communication between vehicle ECUs and diagnostic tools, accurately retrieving DTC (DTCs) and other relevant data. The system's modular and scalable design also supports future expansions for additional diagnostic functions, making it adaptable for evolving automotive technologies. This OBD protocol system stands as a crucial tool for vehicle maintenance, reducing downtime and promoting proactive vehicle care, thereby contributing to improved safety, performance, and environmental compliance in the automotive industry.

REFERENCES

[1] R. Malekian, N. R. Moloisane, L. Nair, B. Maharaj, and U. A. K. Chude-Okonkwo, "Design and Implementation of a Wireless OBD II Fleet Management System," IEEE Transactions on Intelligent Transportation Systems, vol. 18, no. 10, pp. 1–10, 2017.

[2] S.-h. Baek and J.-W. Jang, "Implementation of Integrated OBD-II Connector with External Network," Information Systems, Elsevier, 2014. DOI: 10.1016/j.is.2014.06.011

[3] P. C. Joseph and S. P. Kumar, "Design and Development of OBD-II Compliant Driver Information System," Indian Journal of Science and Technology, vol. 8, no. 21, pp. 1–8, Sep. 2015. DOI: 10.17485/ijst/2015/v8i21/79516

[4] J.-S. Jhou, S.-H. Chen, W.-D. Tsay, and M.-C. Lai, "The Implementation of OBD-II Vehicle Diagnosis System Integrated with Cloud Computation Technology," in Proc. of 2nd Int. Conf. on Robot, Vision and Signal Processing (RVSP), 2013, pp. 267–270. DOI: 10.1109/RVSP.2013.55

[5] J. V. Moniaga, S. R. Manalu, D. A. Hadipurnawan, and F. Sahidi, "Diagnostics Vehicle's Condition Using OBD-II and Raspberry Pi Technology: Study Literature," Journal of Physics: Conference Series, vol. 978, no. 1, p. 012011, 2018. DOI: 10.1088/1742-6596/978/1/012011

[6] M. M. Oluwaseyi and M. S. Abolarin, "Specifications and Analysis of Digitized Diagnostics of Automobiles: A Case Study of On-Board Diagnostic (OBD II)," International Journal of Engineering Research & Technology (IJERT), vol. 9, no. 1, pp. 91–96, Jan. 2020.

[7] ISO 15765-4:2016 – Road vehicles — Diagnostic communication over Controller Area Network (DoCAN) — Part 4: Requirements for emissions-related systems.

[8] ISO 15031-5:2015 / SAE J1979 – Communication between vehicle and external test equipment for emissions-related diagnostics.

[9] ISO 15031-3:2004 / SAE J1962 – Diagnostic connector and associated electrical interface requirements.

[10] ISO 15765-2:2016 – Transport protocol and network layer services for CAN (ISO-TP used for multi-frame CAN messaging in OBD).