**OBD - SCANNER**

**INTRODUCTION:**

**OBD = On-board diagnostics:**

It monitors and regulates a vehicle’s performance by including a network of sensors that collect information which is inbuilt in modern car’s systems and alert the driver to get rid of problems.

**OBD-1 SCANNER:**

1. Monitored basic engine functions.

2. Drawback: No standard, limited data, manufacturer-specific tools.

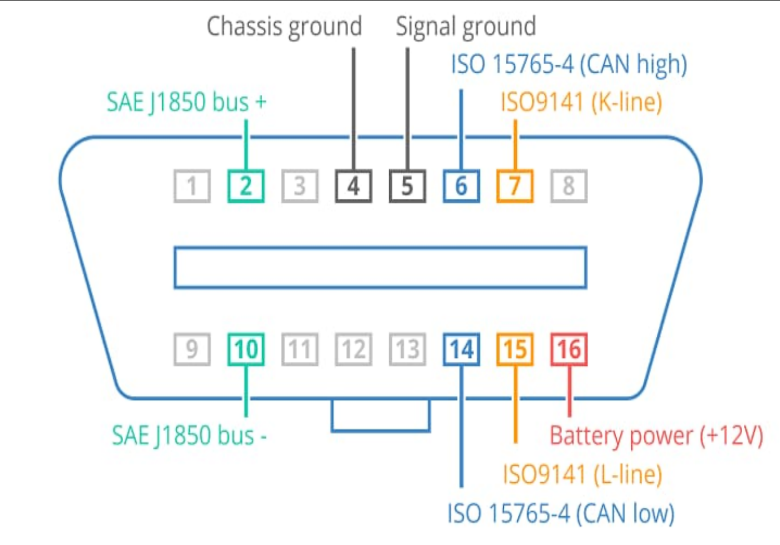
**OBD-II SCANNER:**

1. Standardized system, more detailed data.

2. Improvement: Universal codes, real-time monitoring, and broader diagnostics. It can use wireless connections (via Bluetooth or Wi-Fi) for diagnostics. This allows easy real-time monitoring on mobile apps without needing a physical cable. OBD-I lacks this feature due to its limited technology.

# 1. HARDWARE INTERFACE DESIGN:

## **OBD-II Connector Design:**



A CAN (Controller Area Network) prototype in cars refers to a development model or testing version of a vehicle's electronic control system. CAN is a standardized network protocol that allows microcontrollers and devices to communicate with each other within a vehicle. It enables data exchange between various electronic control units (ECUs), such as:

* Engine control unit (ECU)
* Transmission control unit (TCU)
* Anti-lock braking system (ABS)
* Airbag control unit (ACU)

**CAN pin configuration:**

The OBD-II port has 16 pins, but for basic functions like reading RPM, speed, and Diagnostic Trouble Codes (DTCs),

* Pin 4 & 5: Ground.
* Pin 16: Power (12V).
* Pin 6 & 14: CAN High and CAN Low (for most modern vehicles). The ECU uses these pins to send sensor data to external diagnostic tools.
* Pin 7: K-Line (used in older cars for diagnostics).

**Microcontroller Selection:**

The ESP32 microcontroller is suitable for this task due to its integrated Wi-Fi and Bluetooth capabilities. In contrast, Arduino boards require external modules for these functions, increasing circuit complexity. While Raspberry Pi is a powerful processor capable of handling demanding tasks, it is unnecessary for this project and exceeds the required budget.

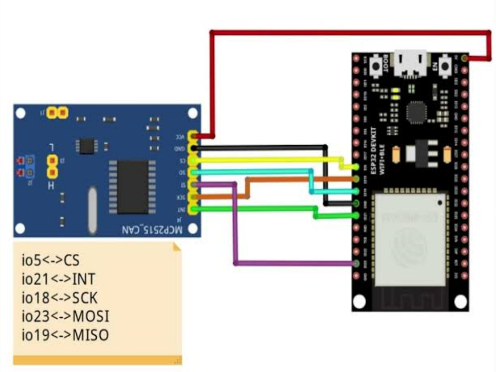
**Additional sensors:** GPS and additionally required sensors can be used according to our basic needs.

**Power supply:** The MCP2551 CAN transceiver converts the 12V DC power supply from the car to 5V or 3.3V DC, making it suitable for the ESP32 microcontroller.

**The CAN transceiver (e.g., MCP2551)**

The CAN transceiver (e.g., MCP2551) is a communication link between the CAN and the esp32. It converts the signals from the car's CAN bus (Pins 6 and 14) into a format that the ESP can read and process. The ESP32 on the shield communicates with the CAN transceiver make the SPI interface.

**Design Diagram:**



## **Hardware Interface Planning:**

A computer screen shot of a computer program

Description automatically generated

A computer chip with a white square

Description automatically generated

A screenshot of a computer

Description automatically generated

* Diagram showing ESP32, MCP2515 and OBD-I
* Use of Wi-Fi/Bluetooth for connectivity with the Blynk app on your mobile device.

2.**FIRMWARE DEVELOPMENT:**

## **Problem Statement:**

Write firmware to monitor engine RPM, vehicle speed, and DTCs using CAN communication.

## **Firmware Code:**

* The firmware communicates with the ECU using CAN protocol, reads RPM and speed, and sends the data to the Blynk app for real-time display, but the existing OBD 2 uses torque app to display the data.
* You can expand it for DTC detection.
* Refer git for the code.

**3. SOFTWARE INTEGRATION:**

* **Data Display:**
* Blynk App: Use widgets to display real-time engine RPM, vehicle speed, and DTCs.
* Code: Use Blynk.run() to keep the app updated with live data.
* **Connectivity:**
* Bluetooth/Wi-Fi: Send data from ESP32 to Blynk app. Use Bluetooth Serial for Bluetooth or Blynk.begin (auth, ssid, pass) for Wi-Fi.
* **Data Logging:**
* Local Storage: Save data on an SD card using the SD.h library.
* Cloud Logging: Use Blynk’s cloud features for remote data storage and analysis.

Use Wi-Fi (via ESP32) to transmit the data to the Blynk app. Alternatively, use Bluetooth if Wi-Fi is not available.

A screenshot of a computer

Description automatically generated.

**CONCLUSION:**

Blynk is great for general IoT projects, while Torque is more suited for real-time OBD-II diagnostics. For OBD-II applications, Torque is recommended, but integrating Torque with Blynk or using compatible libraries can provide the needed functionality for automotive projects. Choosing the right platform ensures success and optimal performance.