**Session - 25**

# Interfacing of CAN modeule with ECU for receiving

Interfacing a CAN module with an ECU (Electronic Control Unit) for receiving data involves connecting the CAN transceiver within the ECU to the CAN bus network, where it listens for and receives messages. The CAN controller within the ECU then processes these messages based on their identifiers and data content

1. CAN Bus Basics:

- CAN (Controller Area Network) is a robust communication protocol used in vehicles and other embedded systems to allow multiple ECUs to communicate with each other without a central host computer.
- It uses a two-wire differential bus (CAN high and CAN low) for reliable data transmission.
- All ECUs on the bus receive all messages, but each ECU filters and processes only the messages relevant to its function based on their message identifiers (IDs).

2. ECU Components for CAN Communication:

- **CAN Transceiver:**

Converts the digital signals from the ECU to differential signals suitable for transmission over the CAN bus and vice versa.

- **CAN Controller:**

Part of the ECU's microcontroller, responsible for higher-level protocol tasks like message filtering, arbitration, error detection, and framing.

- **Microcontroller:**

The central processing unit of the ECU, which uses the CAN controller to interpret received messages and determine appropriate actions

3. Interfacing Steps:

1. **1. Connection:**

The CAN transceiver within the ECU is physically connected to the CAN bus lines (CAN high and CAN low).

2. **2. Configuration:**

The CAN controller within the ECU needs to be configured with the correct settings for the specific CAN network, including bit rate, message identifiers to listen for, and error handling.

3. **3. Message Reception:**

When a CAN message is transmitted on the bus, the transceiver receives the differential signal and converts it to a digital signal for the CAN controller.

4. **4. Message Filtering and Processing:**

The CAN controller examines the message's ID and, if it matches a configured identifier, extracts the data from the message.

5. **5. Action:**

Based on the received data, the ECU's microcontroller can perform specific actions, such as updating sensor readings, controlling actuators, or triggering other events.

4. Key Considerations:

- **Message Filtering:**

ECUs only process messages intended for them, using their unique identifiers.

- **Error Detection:**

The CAN protocol includes robust error detection mechanisms to ensure data integrity.

- **Flexibility:**

CAN's message-based communication allows for easy addition or removal of ECUs and modification of communication flows

**Interfacing a CAN Module with an ECU for Receiving Messages**

**Controller Area Network (CAN)** is a robust communication protocol used in automotive and industrial applications. It allows **Electronic Control Units (ECUs)** to communicate over a shared bus.

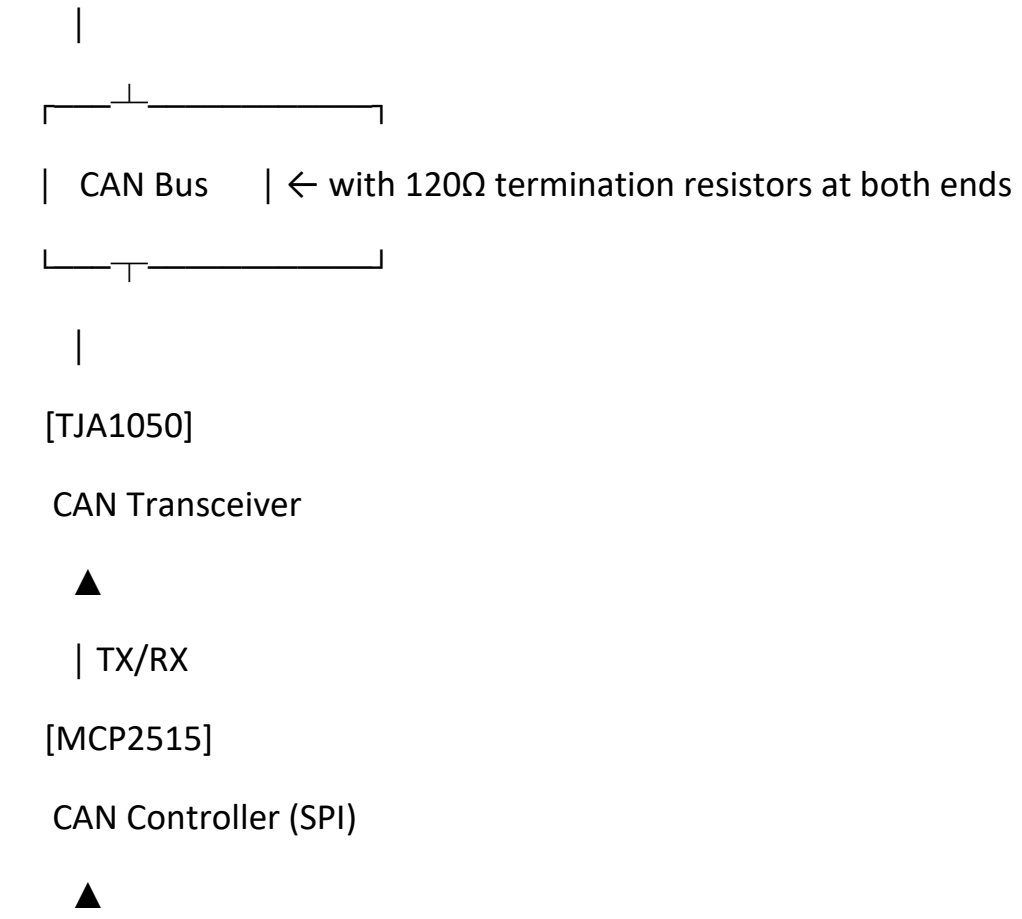
To **receive CAN messages** from an ECU, you typically interface a **CAN transceiver module** with a **microcontroller**, which reads data sent by the ECU over the CAN bus.

🧰 **Basic Components Needed**

| **Component** | **Description** |
|---|---|
| Microcontroller | Arduino, STM32, ESP32, or any with SPI/I2C |
| CAN Controller | e.g., MCP2515 (uses SPI to interface with MCU) |
| CAN Transceiver | e.g., TJA1050, SN65HVD230 – converts logic to CAN bus levels |
| ECU | Sends CAN messages |
| CAN Bus | Physical layer connection (CAN_H, CAN_L) |
| Termination Resistors | 120Ω at both ends of the bus |

## 🔌 Hardware Connections

### 🛠️ Common Setup with MCP2515 (CAN Controller) + TJA1050 (Transceiver):




## 👷 Example: Receiving CAN Messages Using Arduino + MCP2515

### 📦 Libraries:

Install **MCP_CAN** library (for MCP2515):

- Arduino IDE → Library Manager → Search for "MCP_CAN" and install.

**Arduino Code (CAN Receiver):**

```
#include <SPI.h>
#include "mcp_can.h"

const int SPI_CS_PIN = 10;
MCP_CAN CAN(SPI_CS_PIN);

void setup() {
  Serial.begin(115200);
  while (CAN_OK != CAN.begin(CAN_500KBPS)) {
    Serial.println("CAN BUS Init Failed");
    delay(100);
  }
  Serial.println("CAN BUS Init Success");
}

void loop() {
  long unsigned int rxId;
  unsigned char len = 0;
  unsigned char rxBuf[8];

  if (CAN_MSGAVAIL == CAN.checkReceive()) {
```

```
    CAN.readMsgBuf(&rxId, &len, rxBuf);


    Serial.print("ID: ");
    Serial.print(rxId, HEX);
    Serial.print(" Data: ");
    for (int i = 0; i < len; i++) {
      Serial.print(rxBuf[i], HEX);
      Serial.print(" ");
    }
    Serial.println();
  }
}
```

### 📶 CAN Message Frame

| Field | Description |
|---|---|
| **ID** | Message identifier (e.g., 0x100) |
| **DLC** | Data Length Code (0–8 bytes) |
| **DATA** | Up to 8 bytes per frame |
| **CRC/ACK** | Handled by hardware layer |

## ✅ Checklist for Receiving from ECU

1. **Wiring:**
    - Connect **CAN_H** and **CAN_L** from ECU to MCP2515 CAN transceiver.
    - Ensure 120Ω termination resistors on both ends of the CAN bus.
2. **Baud Rate:**
    - Set your MCP2515 to the **same baud rate** as the ECU (usually 250 kbps or 500 kbps).
3. **Message ID:**
    - Make sure you're looking for the **correct CAN ID** being sent from the ECU.
4. **Termination:**
    - Verify the bus is properly terminated to avoid reflections.

## 🧪 Debug Tips

- Use a **CAN analyzer** tool (like USB-CAN adapter) to verify traffic from ECU.
- Use an oscilloscope to confirm signals on CAN_H and CAN_L.
- Try a **loopback mode** test on MCP2515 to check if the CAN module is working.