Session - 27

ECU Design for Dashboard Panel

Designing an **Electronic Control Unit (ECU)** for a **car dashboard panel** involves creating a microcontroller-based system that monitors vehicle parameters (speed, fuel level, RPM, temperature, etc.) and drives output devices like displays, indicators, and gauges.

📘 THEORY: ECU for Dashboard Panel

✅ Functions of Dashboard ECU:

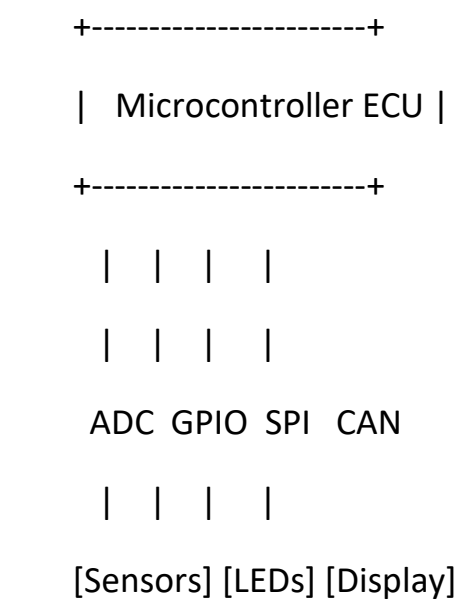
- Read sensors (speed, fuel, temp, RPM, etc.)
- Display data on:
    - Digital screen (LCD/TFT/OLED)
    - Analog meters (using PWM/servo)
    - LEDs (warning indicators)
- Communicate with other ECUs via **CAN bus**
- Handle alerts (engine warning, fuel low, door open)

🧩 Key Components:

Component	Description
Microcontroller	Central ECU (e.g., STM32, Atmega, ESP32, etc.)
Sensors	Speed, RPM, Fuel level, Temp sensors (analog/digital)
Display Unit	LCD, OLED, or TFT to show status

Component	Description
CAN Transceiver	Communicates with vehicle's other ECUs
Input/Output Drivers	For controlling buzzers, LEDs, motors

🖧 System Block Diagram:



📡 Sensor Inputs (Examples)

Sensor	Interface	Purpose
Speed Sensor	Pulse/Analog	Show vehicle speed
Fuel Sensor	Analog	Show fuel level
RPM Sensor	Pulse	Show engine RPM
Temp Sensor	Analog	Show engine temp

👷 SAMPLE PROGRAM (Arduino-based ECU Simulation)

This simple simulation:

- Reads dummy sensor values
- Displays data on Serial Monitor (you can replace this with an actual LCD)
- Sends data over **CAN** to another ECU

✅ Requirements:

- Arduino Uno + MCP2515 CAN Module
- Simulated analog inputs for speed, temp, fuel, RPM

✅ Arduino ECU Code (Sender)

```
#include <SPI.h>

#include "mcp_can.h"


const int SPI_CS_PIN = 10;

MCP_CAN CAN(SPI_CS_PIN);    // Set CS pin


void setup() {

  Serial.begin(115200);

  while (CAN_OK != CAN.begin(CAN_500KBPS)) {

    Serial.println("CAN Init Failed");
```

```
    delay(100);
  }
  Serial.println("CAN Init Success");
}

void loop() {
  byte data[8];

  // Simulated analog sensor values (0–255)
  data[0] = analogRead(A0) / 4; // Speed
  data[1] = analogRead(A1) / 4; // RPM
  data[2] = analogRead(A2) / 4; // Temp
  data[3] = analogRead(A3) / 4; // Fuel

  CAN.sendMsgBuf(0x100, 0, 8, data); // ID 0x100 for dashboard data

  Serial.print("Speed: "); Serial.print(data[0]);
  Serial.print(" RPM: "); Serial.print(data[1]);
  Serial.print(" Temp: "); Serial.print(data[2]);
  Serial.print(" Fuel: "); Serial.println(data[3]);

  delay(1000);
}
```

✅ **Arduino Dashboard Code (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 Init Failed");
    delay(100);
  }
  Serial.println("CAN Init Success");
}

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

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

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

    if (rxId == 0x100) { // Match dashboard ECU data
      Serial.print("Speed: "); Serial.print(buf[0]);
      Serial.print(" RPM: "); Serial.print(buf[1]);
      Serial.print(" Temp: "); Serial.print(buf[2]);
      Serial.print(" Fuel: "); Serial.println(buf[3]);
    }
  }
}
```

🧠 Real-World Enhancements

To move from simulation to real vehicle use:

- Replace analog inputs with actual vehicle sensor interfaces
- Use **CAN DBC file** for decoding messages if required
- Drive **TFT or OLED displays** using SPI or I2C
- Add **warning lights or buzzers** based on threshold logic
- Add **EEPROM or SD card logging**

📘 Summary

Designing a **dashboard ECU** involves:

- Interfacing sensors to monitor parameters
- Processing and formatting data
- Communicating via CAN to other ECUs
- Displaying data to the driver in real-time