Session 22

Interfacing of CAN module with ECU for sending

Interfacing a CAN module with an ECU for sending messages

Interfacing a CAN (Controller Area Network) module with an ECU (Electronic Control Unit) to send messages involves connecting the physical hardware and configuring the software to facilitate communication via the CAN bus protocol.

Here's a general overview of the process:

1. Understand the CAN bus basics

- CAN is a serial communication protocol that enables ECUs in vehicles to communicate without a host computer.
- It operates on a two-wire bus (CAN High and CAN Low) using differential signaling for robustness against noise.
- Messages are transmitted in frames with a unique identifier that also determines message priority (lower ID = higher priority).
- CAN ensures data integrity through error detection mechanisms and handles simultaneous transmissions through arbitration.
- There are different CAN standards (e.g., CAN 2.0, CAN FD, CAN XL) offering varying data rates and capabilities.

2. Identify the ECU's CAN interface

- An ECU that needs to participate in CAN communication will have a built-in CAN interface.
- This interface consists of a CAN controller and a CAN transceiver.
- The CAN controller handles protocol-specific functions like message framing and error detection.
- The CAN transceiver connects the controller to the physical CAN wires and handles signal conversion and electrical protection.

3. Choose and connect the CAN module (if necessary)

- If your ECU doesn't have an integrated CAN interface, you can use an external CAN module, such as an MCP2515 CAN module paired with an Arduino board, as an example.
- This module would be interfaced with your ECU (e.g., microcontroller) via SPI communication.
- Ensure correct pin connections between the CAN module and the ECU (VCC, GND, CS, SO, SI, SCK, INT).

4. Physical connection to the CAN bus

- The CAN module's CAN_H and CAN_L pins are connected to the corresponding lines of the CAN bus.
- In a multi-node setup, all nodes (ECUs and any external modules) are connected to the same CAN_H and CAN_L lines.
- Termination resistors (typically 120 Ω) are placed at each end of the CAN bus to prevent signal reflections.

Sample Pseudocode

```
#include <stdint.h>


// CAN message structure
typedef struct {
    uint32_t id;       // Standard or Extended ID
    uint8_t dlc;       // Data length (0-8)
    uint8_t data[8];   // Data payload
} CAN_Message;
```

```
// Initialize CAN Controller (baud rate, filters, interrupts)
void CAN_Init(void) {
    // Set CAN baud rate (e.g., 500 kbps)
    // Configure CAN filters (accept all for simplicity)
    // Enable CAN controller and interrupts
}

// Send CAN message
int CAN_SendMessage(CAN_Message *msg) {
    // Wait for transmit mailbox to be free
    // Load msg->id, msg->dlc, msg->data into CAN hardware registers
    // Request transmission
    // Return success or failure status
    return 0; // success
}

// Example usage
int main(void) {
    CAN_Init();

    CAN_Message msg;
    msg.id = 0x123;        // Standard ID
    msg.dlc = 4;           // 4 bytes of data
    msg.data[0] = 0xDE;
```

```
    msg.data[1] = 0xAD;
    msg.data[2] = 0xBE;
    msg.data[3] = 0xEF;

    if (CAN_SendMessage(&msg) == 0) {
        // Message sent successfully
    } else {
        // Handle send failure
    }

    while(1) {
        // Main loop
    }
}
```