AMS Project Report - Task 2

Overview

This document outlines what I did for Task 2 of the AMS improvements project, which I completed as part of my application to join RMIT Motorsport. The aim of the task was to transmit the minimum and maximum cell voltages and temperatures from the AMS unit to other modules on the CAN network. To do this, I built on top of the existing R24_AMS CANBUS system, which already handles message formatting and scheduling.

```
void CANBUS_SEND_AMS_MINMAX_VT(CAN_HandleTypeDef *hcan, CAN_TxHeaderTypeDef *TxHeader, Car_State_Handle *state)
{
    txHeader->StdId = MSGID_AMS_MINMAX_VT;
    txHeader->dlc = 8;      You, last week • Uncommitted changes
```

What I Needed to Send

The message included:

- The lowest cell voltage
- The highest cell voltage
- The lowest cell temperature
- The highest cell temperature

All of that had to fit in a single 8-byte CAN message, so I had to scale the values properly.

What I Changed in the Code

In canbus_msgs.c:

- I added a new function: CANBUS_SEND_AMS_MINMAX_VT()
- It grabs the latest readings using bms_get_measurements()
- Then it works out the min and max values and scales them:
- Voltages go to millivolts (mV)
- Temperatures are converted to tenths of degrees (0.1°C)
- After that, the values are packed into an 8-byte array and sent using CANBUS_BASE_SEND_MSG()

In canbus_msgs.h and canbus.h:

- I added the function declaration
- Assigned a new CAN message ID: 0x602, labeled as MSGID_AMS_MINMAX_VT

In canbus.c:

- I set the message up to be sent periodically from the AMS node

In main.c:

- Double-checked that the AMS node was initialized properly

- Confirmed the CANBUS task runs every 1ms inside the main loop

Testing & Results

Unfortunately, I couldn't test this on hardware because:

- I didn't have access to the car or ECU at the time.
- I also didn't have a working CAN logging setup.

Everything was developed and compiled using STM32CubeIDE. If I were to test this in the future this is how I would go about it:

Flash the AMS firmware:

- Load the firmware using an ST-Link.

Set up a CAN network:

- Connect to a test CAN bus using something like CANalyzer, CANoe, or a USB-to-CAN adapter.

Check message traffic:

- Look for messages with ID 0x602 being sent at 50Hz (every 20ms).
- Confirm the data layout:
  - Bytes 0 to 1: Min Voltage (mV)
  - Bytes 2 to 3: Max Voltage (mV)
  - Bytes 4 to 5: Min Temp (dC)
  - Bytes 6 to 7: Max Temp (dC)

Simulate edge cases:

- Manually adjust the cell_voltage[] and cell_temperature[] arrays to test different values.

Optional UART Debugging:

- Print values before sending to help with debugging and validation.

Conclusion

Even though I didn't get to test the code on the actual car, I followed the team's CANBUS framework to the best of my ability. The feature compiles successfully and runs in simulation. It's ready for hardware testing when the setup becomes available.