J1939 Project Tracking

Completed Work

* **Concept**
  + Created the concept of the project with a project requirements document.
* **Hardware/Software Selection**
  + Selected all the hardware needed and software required, along with creating the schematic.
    - This process was a bit of trial and error, time was spent trying to vet out different options in a cost-effective way to accomplish the goals outlined in the requirements.
* **Configuration**
  + Setup and configured the STM32 with the Cubemx IDE, involving all the SPI, GPIO, and clock settings to best work with the MCP2515.
* **MCP2515 Drivers**
  + Wrote wrapper functions for the low-level HAL functions provided by STM32
  + Wrote various other SPI drivers to work with the MCP2515 which include reading and writing sequences specific to this chip.
  + Added many defines to the project for the various different register locations on the MCP2515 for readability purposes.
  + This includes time spent reading the datasheet for the MCP2515 to understand how to write these drivers, having never worked with a chip like this before.
* **Debugging**
  + Much of the debug time was spent debugging the MCP2515 drivers, as well as weird behavior that was attributed to some settings in the STM32 configuration and potentially bugs within the STM32 IDE software, hard to say exactly what my issue was.
  + Spent time creating testing and output functions to clearly see what is going on with the data, and what values were in the MCP2515 registers.

Work to Complete

* **MCP2515 Configuration**
  + The last task with the MCP2515 will be to configure the registers so that the output is 250k/bits.
  + This is not a trivial task and will take time looking at the datasheet to fully understand how to do this, although my mentor mentioned he has a very solid understanding and has used this chip before.
* **J1939 Library Functions**
  + I don’t yet have a full grasp of what is required here. As of now my drivers handle all communication and are able to take the full CAN message and break that down into the 29-bit identifier, dlc, and breakout all the data bytes individually.
  + At the very least I would need to create a system to store all incoming messages that I want to read and process them so that my application code can easily use them to determine what to do next.
  + I am undecided if I want to process DTC’s, it was in my original outline, but this will take more time.
    - Same with transport protocol.
* **Application Code Instructions**
  + The main concept of this project outlined that the user would need to create a .dbc file based on the data given to them and they will need to send messages to the module. The module in turn would respond to the CAN message, as well as printing out debug messages to the serial port to inform them if the message was valid or not.
  + The application code for this should be straight forward and hardcoded to respond or not respond to certain PGNs and data values.
* **Instructions/Outline**
  + The instructions will need to be written so that the user can follow them and correctly use the module. This part is subject to change and will be thought about more when I’m writing them. As of now they will include:
    - Instructions for how to setup the module, including both the hardware setup and the software they will need to send CAN messages.
    - All of the data that would normally be included inside of a .dbc file.
      * They will need to input all this data into a .dbc file to get familiar with that type of file.
    - Instructions on how to process the data that is received back.
    - Maybe some sort of troubleshooting section.
* **Debugging**
  + There will inevitably be more debugging involved with everything and time should be accounted for to some degree. I have the controller working for the most part, although I have not sent or received legitimate CAN messages to my PC yet, that will still need to be tested.