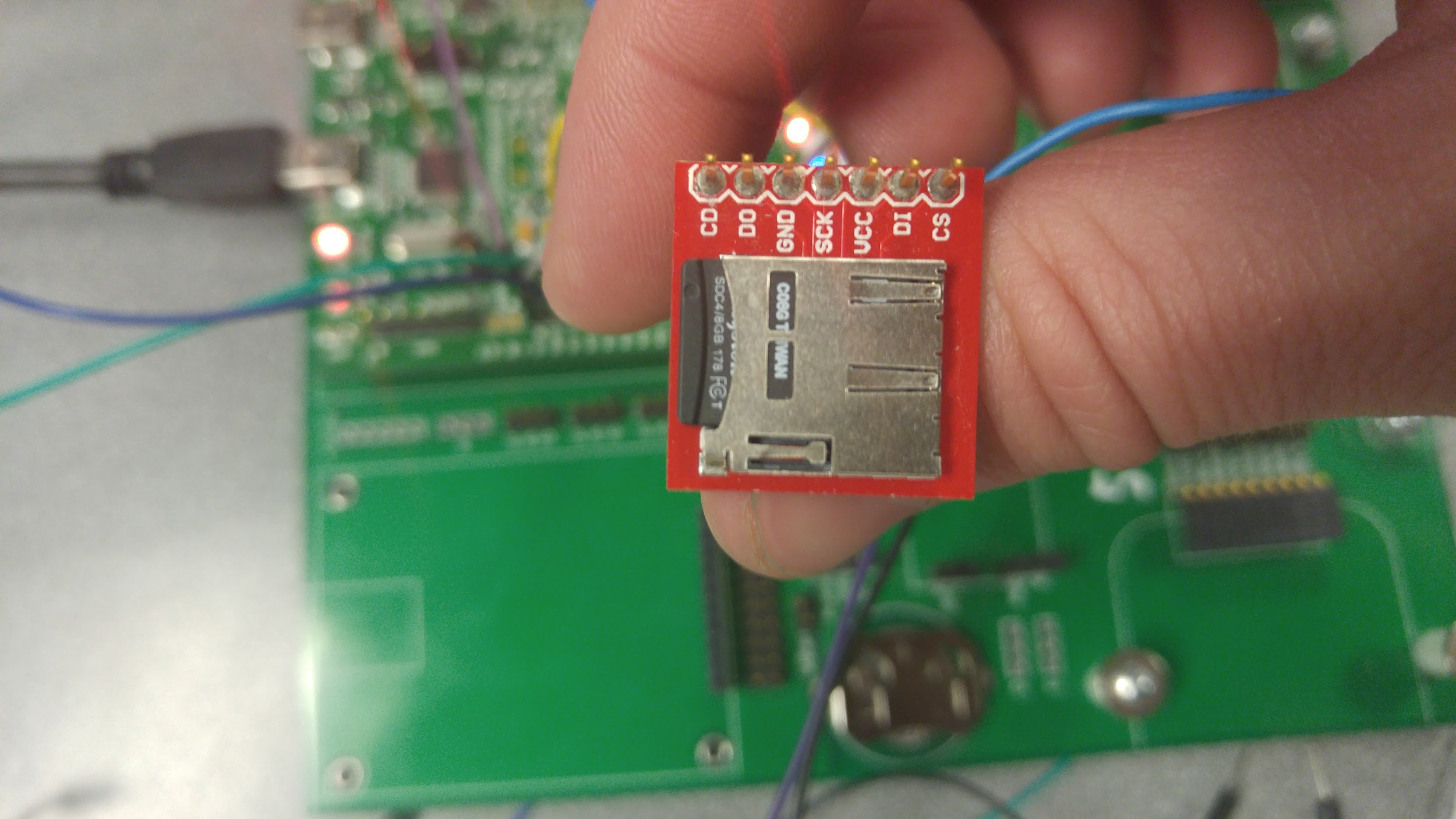
Power Milestone 3

Colin Thackston, Geri Rista, Jake Rogers

For this milestone, our task was to create a system that logs power data to the SD card while also recording the current state of the system. This is an extension of the previous milestone where we logged power using an STM32-based system. We relied on the same set up using the μCurrent device to log power and then storing that in a large buffer. However, we then push that buffer to the SD card for additional storage. By not allowing Colin to access any of the repositories, we managed to avoid the github issues that have plagued us for the previous two milestones.

As with last time, our setup requires two boards. One board logs power and writes to the SD card and the other board runs code (our substitute for the trailCounter) and writes to GPIO pins. There are wires connecting the two boards’ GPIO pins and the boards are also linked through the μCurrent device which produces a voltage. Using the same ADC code from milestone 2, we convert the reading from the μCurrent to a number where it is stored in RAM. To record the state of the code, we simply set a flag which changes a global variable when the program enters a new state. For example, one is running, two sleep, and three is standby. These states are then written to GPIO pins and the other board reads the setting. For this, we recycled code from the pressure sensor lab and some built in functions from ChibiOS such as palSetPad, palClearPad, and palReadPad. These high level abstractions saved us from having to initialize pins and write any driver files. We write to the SD card at a much slower rate than the sensor group did because we quite simply do not need as much data. We only need to know when the program changes states and when threads run for too long. We do not need gigabytes of data to reconstruct that picture.



Utilizing the SD card proved to be much more difficult than we expected. The sensor group already completed the arduous process of writing the code for writing to the SD card which saved us a substantial amount of man hours. Configuring the ChibiOS files, drivers, and the Makefile even after we received the SD card code consumed much of our time. To fix the SD card, we had to update all of our driver files so they would correctly utilize SPI. We also ran into several self-inflicted problems caused by our inability to copy and paste correctly. We had to create new paths to our ChibiOS files for compilation and change several of the driver file settings to enable the SPI connection to work across the SD card. Writing the state to the GPIO pins and then reading it back did not produce many difficulties to us. Originally, we used code from the LED drivers to set the pins which did work, however, it was quite cumbersome. We ended up just using the built-in ChibiOS functions which gave us far more functionality.

As for future tasks, we want to make a ChibiOS shell interface for our program. Currently, we just have several chunks of code that we periodically comment out in order to run the pieces we want. This will make our program easier to use for anyone who did not participate in the software development. The other additional feature we wish to add is another GPIO pin that will return the current thread running. Right now, our program logs the current state and the voltage. While useful, this approach provides us with very little attribution. Our plan for the trailCounter is to make each thread set the value of a global variable which we will push to a third GPIO pin. This way we will be able to tell not only how much power is being used, but also what is using the most power and if the code is getting stuck somewhere it should not be. We anticipate this being a major asset for debugging the device when we actually deploy it into the field.

