Garrett Burton & Wendy Kwok

Milestone II Report

Introduction

The purpose of this milestone is to set up firmware that will be able to take data for both RTC time and GPS time. We will be using the GPS puck, which should have accurate timing, to compare to the RTC clock time. By doing so, we find the differences and will be able to analyze the drift with it. Along with the drift, we have to collect temperature data because depending on temperature, the oscillator will change its frequency slightly, causing the time to be different. So, the temperature at time t should tell us how much the drift is for that specific temperature.

Setting up the GPS and RTC time collection

The GPS is set up with a with a wire soldered to a pulse based off of the GPS's led light. As soon as the GPS find's a satellite and locks into the signal, its led will start blinking once per second. This pulse is connected to one of the STM32's pins. We use this pulse as an interrupt, signalling a semaphore on a thread waiting to collect data. Once the signal is released, the RTC time, GPS time, and temperature is recorded onto an SD card. The SD card code was provided by a group that has already worked on the read/write SD. A difficulty of this part was that initially we did not know there existed a pulse per second, and spent some time wondering how to accurately capture the data. With the pulse per second, we were able to capture data on a specific edge, making it more accurate. Another difficulty that presented itself was that the GPS puck did not receive signal or cannot lock onto a satellite signal easily. This made it problematic to test out since the data collection relied on a pulse connection. As a temporary, we used the STM32's user button to test as a signal for the interrupt.

Parsing GPS constant data

In order to collect time data from the GPS, we need the GPS to find a signal. Once it finds signal it is able to keep track of the time. How the GPS provides data is that it gives you information on time and location based on the NMEA sentence formats. Each sentence is started by a sentence tag, and then information separated by commas. Usually, what each tag means are explained in documentation. For our milestone, we needed to just parse out one sentence, where at the beginning of the sentence was the time we needed. So far, it seems, it is very difficult to find a way to parse stream of data quickly, and the time seems to update every 6 seconds, which is a 10th of a minute. We had lots of difficulty trying to get the buffer size to be bigger, so that we can incorporate more data in one go.

Setting up the temperature collection:

For this milestone we were required to get the temperature sensor reading a temperature and stored along our other required data. This proved to be more difficult than we had first anticipated. We were lucky enough to use code provided by Colin and the power team allowing us to set up the ADC, which controls the onboard temperature sensor. To read the temperature we needed to receive the signal coming from channel 16, this is where we came into a roadblock. Although Colin's code successfully initialized the ADC it was not configured to allow reading from channel 16. We will continue working on this with the help of Bryce and are hoping to have it reading by

Monday of next week so we can collected 3 consecutive days of data for our Milestone 3 requirement. Here’s a look at the initialization of the ADC structure:

```
StaticconstADCConversionGroupadcgrpcfg={
FALSE,
1,
NULL,
NULL,
0,
0,
ADC_CCR_VREFEN|ADC_CCR_TSEND,
{0,
ADC_SMPR@_AN16(ADC_SMPR_SMP_601P5)},
{ADC_SQR1_NUM_CH(1)|ADC_SQR1_SQ1_N(ADC_CHANNEL_IN16),
0,
0,
0},};
```

The reading from the temperature sensor does not return a number in Celsius or Fahrenheit, but rather a representative number that needs to be converted to a standard temperature unit in our case Celsius. The algorithm needed to do this is present in the lab manual, here’s a look at it:

```
temperature = (int)((V25 - ((3.3 * samples_buf[0])/ 4096.0))/Avg_Slope + 25);
```

V25 represents the base at 25 degrees. The temperature as we have it currently set up is pulled within our time thread. We pull and convert the temperature as we pull the time making sure we have an accurate representation of temperature at the exact moment we take the time, which will later allow us to develop an algorithm with the most certainty.

Conclusion

What we have at the end of this week is not exactly finished. The GPS data is parsing but it is not updating as fast as we’d like it. The temperature is calculated correctly but it seems like it is offset by some base temperature that needs calibration. We have taken a look at the documentation and even searched online for any data on the calibration, but it seems that ST only says it is calibrated at the factory. We hope to find something that maybe calibrates the temperature into a more accurate one.