

Washington University in St. Louis

Washington University Open Scholarship

Mechanical Engineering and Materials Science
Independent Study

Mechanical Engineering & Materials Science

5-16-2023

Data Collection Research for Multidisciplinary Design & Prototyping Fall 2023: Spring 2023 MEMS 400 Independent Study

Hayato Takai

Washington University in St. Louis

Julian Wu

Washington University in St. Louis

Follow this and additional works at: <https://openscholarship.wustl.edu/mems500>

Recommended Citation

Takai, Hayato and Wu, Julian, "Data Collection Research for Multidisciplinary Design & Prototyping Fall 2023: Spring 2023 MEMS 400 Independent Study" (2023). *Mechanical Engineering and Materials Science Independent Study*. 238.

<https://openscholarship.wustl.edu/mems500/238>

This Final Report is brought to you for free and open access by the Mechanical Engineering & Materials Science at Washington University Open Scholarship. It has been accepted for inclusion in Mechanical Engineering and Materials Science Independent Study by an authorized administrator of Washington University Open Scholarship. For more information, please contact digital@wumail.wustl.edu.



Spring 2023 MEMS 400 Independent Study

Advisors: Jackson Potter

Report Submission Date: Thursday, May 9, 2023

We hereby certify that the lab report herein is our original academic work, completed in accordance with the McKelvey School of Engineering and Undergraduate Student academic integrity policies, and submitted to fulfill the requirements of this assignment:

Julian Wu



Hayato Takai



Table of Contents

Abstract	3
Introduction	3
Electronics:	5
I2C Serial Communication	5
Integrating with Arduino	6
Finding I2C Devices with Arduino	6
I2C over long Distances	6
I2C Modules	7
Real Time Clock (RTC)	7
SD card	8
Temperature	9
Magnetic Encoder	10
Optical Encoder	11
Development Board	12
Printed Circuit Board (PCB)	12
Schematic	13
Board Comparison	14
Skills	15
Connections	15
Logic Level Shifting	15
General:	15
Data Transfer	16
Bluetooth	16
Pogo Pins:	16
Deep Sleep	16
Prototyping	18
CAD (Fusion 360)	18
Laser Cutting	18
3D printing	18
Hot glue	18
Weather Station Design:	19
Radiello Cover Design	26
Code	27
I2C Data Logging Sample Code	27
Bluetooth Serial Transmit from Sd Card	33

Abstract

Dr. Jackson Potter is a Mechanical Engineer and professor at Washington University in St. Louis. One of the classes he teaches is called Multidisciplinary Design & Prototyping (MEMS 312). The class is taken by students from various disciplines to learn about design processes and develop new skills. In addition, this class typically has a theme to make the prototyping process more realistic and have an end goal. For 2022 the theme was accessible gaming, where students would create universal gaming controllers for people that might not be able to use traditional controllers. For the next year, Dr. Potter wanted to focus on sensors, so the theme focuses on creating robust designs that could house sensors and be functional outside, teaching students about the different sensors and prototyping the housing for it. Additionally, a real life application is that an air quality monitoring system needs to be stored and deployed efficiently, so we tried to come up with a minimum viable prototype to address the issue. This document is for compiling all potential content for the Fall 2023 course theme.

Introduction

Weather Kit:

Environmental sensors are used all over the world to store and transmit data. It's important to have data covering multiple areas so scientists and researchers can get a good idea about how weather patterns change. Some areas are pretty remote and can't be accessed very easily, so some setups need to be robust enough to last long periods of time before being replaced or taken down. Some characteristics of weather stations are that they are weatherproof, have long-lasting batteries, can transmit data over long distances or store data internally, and provide precise measurements.

An example of a weather station that might be used to take data is the [Sparkfun Weather Meter Kit](#).

Figure 1 below shows the Sparkfun Weather Meter Kit with its components.



Figure 1 Sparkfun Weather Meter Kit

This device measures rain, temperature, windspeed, and wind direction, and stores it either on an SD card or directly into a computer. However, this device also comes with its own flaws. Besides having its own interfacing system (not compatible with arduino, and uses drag and drop), the wind vane only measures 8 directions, So if it was pointing let's say at 30 degrees, there would be no direction listed because the switch wasn't activated. The second drawback is that the device itself costs ~\$150 for everything to get it up and running, which costs far more than the cheap sensors used. Therefore, the task for independent study was to come up with a cheaper alternative that can outperform the weather kit in terms of accuracy, precision, battery life, and programmability.

Radiello:

Radiello is an air quality monitoring system that is embedded into a small cylinder and can be taken to very remote areas. It was developed to be light, versatile, inexpensive, and not require any power to be operated. The way it works is that if it senses specific particles in the air, it will change colors and alert the user that there is something in the air. The issue is that sometimes researchers don't want to measure the specific areas that they pass through, so they need an efficient way to store and then quickly deploy the radiello when needed.

Electronics:

I2C Serial Communication

I2C is a serial communication bus that was designed in 1982 by NXP Semiconductors. It is used widely in prototyping due to its ease of use and support. I2C gained popularity because it is a synchronous 2-wire interface, consisting of the Serial Clock Line (SCL) and the Serial Data Line (SDA). The SCL is generated by the controller and synchronizes the signal between the controller and the peripherals. The SDA line is the data signal. In total, you need four wires when you add VCC for power and GND for ground. Another reason I2C is popular is because you can connect up to 112 devices with 7-bit addressing, usually indicated as a hexadecimal value. I2C is used in most sensors nowadays. This is nice because you can add a bunch of sensors on just two lines when previously, they would take up at least one IO pin on the microcontroller.

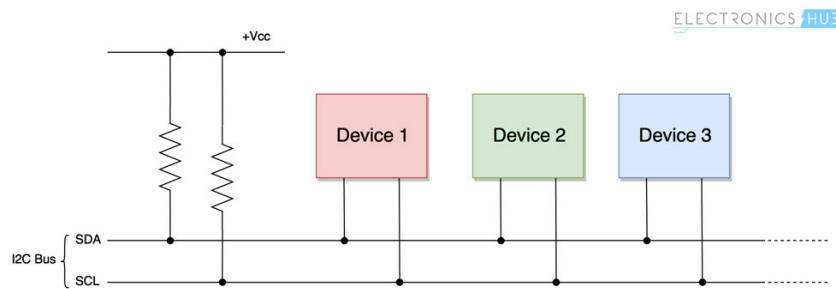


Fig 1. <https://www.electronicshub.org/basics-i2c-communication/>

I2C also frees up processing on the controller side. There is less to worry about when compared to receiving raw input from sensors which often suffer from being very finicky.

Data is transferred in sequences of 8 bits preceded by a start sequence and ends with a stop sequence. The two lines are open drain so pull up resistors (2k for 400kbps and 10k for 100kbps) are needed so that the default for the lines are high (Fig. 1). If the signal is not acknowledged, the controller will decide what to do. These videos are good for understanding conceptually but the actual application will consist of following datasheets and github repositories from the manufacturer. I2C also lets microcontrollers communicate with each other, which can be helpful in some situations.

I2C: <https://youtu.be/6IAkYpmA1DQ>

I2C vs SPI vs UART: <https://youtu.be/IyGwvGzrqp8>

Pull-up Resistors: <https://youtu.be/Ef9CpzipnjQg>

Addressing: <https://youtu.be/8R13KHx4dTQ>

Integrating with Arduino

1. Include the wire library

```
#include <Wire.h>
```

2. The easiest way to use an I2C connection is to find a library for it.

```
Ex. #include <SparkFun_RV8803.h>
```

3. Next, initialize the instance for the device.

```
Ex. RV8803 rtc;
```

4. Start the I2C connection with the wire library

```
Wire.begin();
```

5. Connect to the specific device (The begin function usually returns a boolean, for status)

Ex.

```
    if (rtc.begin() == false){  
Serial.println("Device not found. Please check wiring. Freezing.");  
while (1);  
    }  
Serial.println("RTC online!");
```

Finding I2C Devices with Arduino

The I2C scanner function finds available I2C devices and prints them onto the Serial Monitor. Running this is highly recommended as a first step when troubleshooting.

<https://playground.arduino.cc/Main/I2cScanner/>

I2C over long Distances

Using I2C over long distances (>1m) is not recommended since the signal will suffer.

However if you must, you can lower clock speed to 50 or 10kHz or use a current booster.

Current booster: <https://www.adafruit.com/product/4756>

<https://www.youtube.com/AFRwKN-7NbM>

I2C Modules

*uses I2C

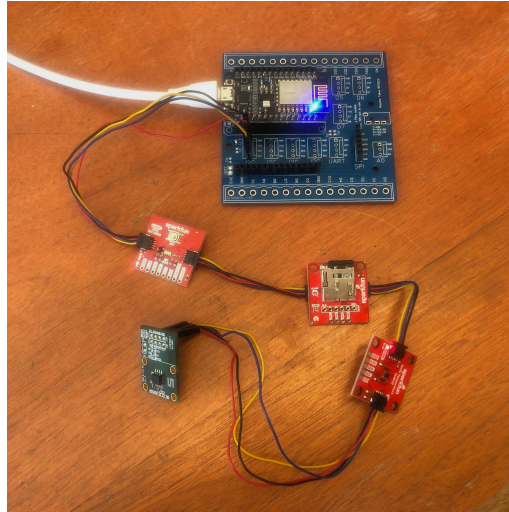


Fig. 2: Development board hooked up to multiple I2C devices

Real Time Clock (RTC)

Description: RTC modules are able to keep track of precise time even when the device is powered off thanks to an external coin battery that keeps the crystal oscillator alive. Features including lower power consumption, wide temperature range, and alarm interrupts make clock modules reliable for data logging applications.

Module tested:

- RV-8803* (\$17.50) <https://www.sparkfun.com/products/16281>

Alternative clock modules:

- DS3231*(~\$3): <https://lastminuteengineers.com/ds3231-rtc-arduino-tutorial/>
- DS1307*(~\$2)

Among the alternatives, DS3231 is recommended over the DS1307 because it is temperature compensated. This means that the timekeeping will not fluctuate with temperature, so it keeps better track of time. The DS1307 has an external oscillator while the DS3231 has an internal one. The DS1307 can end up being off by five minutes per month while the DS3231 only loses a few seconds a month. The DS3231 may be ever so

slightly harder to use than the RV-8803 but would be our first choice when considering cost, availability of tutorials/examples, and features.

SD card

Description: SD card modules are used for storing large volumes of data in a secure location, mostly for data logging applications. They retain their files even when powered off and are directly readable by a computer, making them vital for data logging applications. Most SD card modules use micro SD cards. When logging onto an SD card, it is important to consider what type of delimiter to use when writing the text file. Doing so will allow you to parse the data much easier when opening as a csv. It is also standard to include a line of headers in the text file, which will later become the column header. Another useful tip is giving each file a unique name. This can be done by adding the date/time to the file name. This will prevent newly generated files from overwriting or appending to older files.

Module tested:

- **Openlog Dev-15164*** (\$18.50) <https://www.sparkfun.com/products/15164>

Alternative SD card modules:

- SPI SD card module (~\$3):
<https://lastminuteengineers.com/ds3231-rtc-arduino-tutorial/>

An I2C SD card module was chosen over an SPI (different communication protocol) SD card reader because the SPI connection requires more pins. In our case, we would not have any digital pins left on the ESP8266 which would become a problem when connecting the optical encoder which requires a digital input pin. The I2C module is also easier to use because it has its own ATmega328 and oscillator. It's basically a second arduino nano. The only downside to this is that it can draw up to 20mA when writing to the SD card. SPI SD card modules are much more available but are very loosely regulated. There are multiple versions of the same general PCB layout floating around but since there is no naming for them, you can't easily tell which model you're getting. Regardless, when using SD cards with microcontrollers, the SD card must be formatted to FAT16 or FAT32 and work best with cards that are 64MB to 32GB. A good formatter

is <https://www.sdcard.org/downloads/formatter/>. Disk Utility on Mac was tested and did not work as reliably.

Temperature

Description: Temperature sensors are typically very easy to use and widely available. They have a small footprint so most modules come in the form of a breakout board. They are lower power consumption which minimizes self heating.

Module tested:

- **TMP117*** (\$14.95) <https://www.sparkfun.com/products/15805>

Alternative temperature modules:

- **TMP102*** (\$5.50): <https://www.sparkfun.com/products/13314>
- **AHT20/21*** (~\$4.50) <https://www.adafruit.com/product/4566>
- **AHT10*** (~\$2)
- **DHT22** (~\$4)
- **DHT11** (~2.00)

The recommended temperature really depends on the needs of the applications. The TMP117 is super high precision but comes at a cost. The TMP and AHT line up uses I2C while the DHTs use analog voltage. There are so many temperature sensors out there with different performance results that it is really hard to recommend the “best” one. The DHTs have been around for a long time and are common since they are cheap and easy to find but are considered more of a toy than a data logging instrument. The AHTs are sort of the newer generation of sensors to the DHTs and include humidity sensing but are far from perfect. There are so many sensors out there and even the same models of sensors have different boards and performance at times. Here is a very interesting forum that tests out a bunch of temperature sensors.

<https://forum.arduino.cc/t/compare-different-i2c-temperature-and-humidity-sensors-sht2x-sht3x-sht85/599609> The forum leads to this project page

<https://wiki.liutyi.info/display/ARDUINO/v9+Sensors+Board>

Magnetic Encoder

Description: A magnetic encoder is a contactless angle measurement device that uses a bi-directional magnet to make friction virtually obsolete. By sticking the magnet onto the bottom of a wind vane, the sensor does not make any contact with mechanical components, unlike when using a potentiometer. Magnetic encoders are becoming more common in applications, even more than optical encoders because they are able to maintain their calibration even when powered off when using a permanent bi-directional magnet.

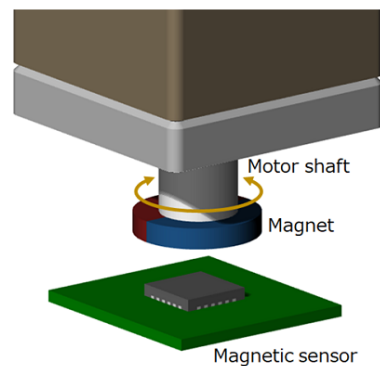


Fig. 3: <https://www.akm.com/us/en/products/rotation-angle-sensor/tutorial/magnetic-encoder/>

Module tested:

- AS5600-SO_EK_AB* (\$16.76)
<https://www.digikey.com/en/products/detail/ams-osram/AS5600-SO-EK-AB/5066879>

Alternative Magnetic Encoders:

- AS5600* (\$3.50):
https://www.amazon.com/Magnetic-Encoder-Induction-Measurement-Precision/dp/B097QNG1CN/ref=sr_1_3?keywords=magnetic%2Bencoder&qid=1683399886&sr=8-3&th=1
- Hall Effect Encoder(~\$8): <https://www.pololu.com/product/3081>

We tested the chip manufacturer, AMS's, official breakout board for the AS5600 chip made by AMS but there are third party boards that use the same chip for much cheaper.

The Hall Effect Encoder is pretty much a deconstructed AS5600 and is not recommended since it does not use I2C.

Other types of Magnetic Sensors:

- Magnetoresistive sensor
- Magneto-impedance sensor

Optical Encoder

Description: Optical Encoders are another type of contactless angle measurement device. They use a code disk with opaque ticks that block out light in pulses between a light source and a photodetector. By increasing the number of ticks on the code disk, the optical encoder is more precise than magnetic encoders but must be calibrated each time the system is reset or if the disk falls out of calibration. It is easier to use optical encoders for measuring rotary speed than for angular position.

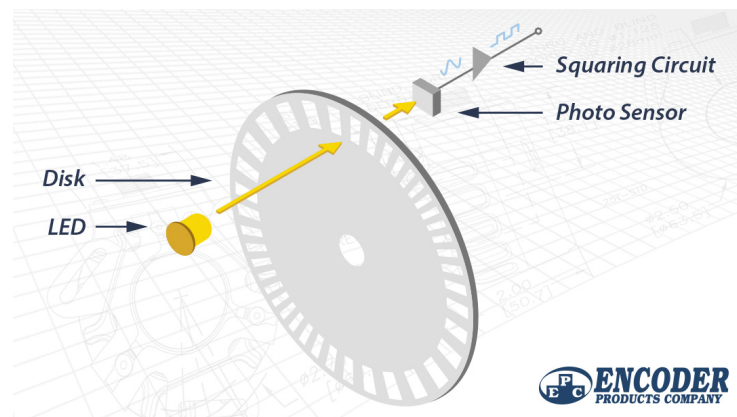


Fig. 4: <https://www.encoder.com/article-what-is-an-optical-encoder>

Development Board

A dev board was created to make testing various communication interfaces and connecting and removing devices easier. On the outermost row, screw block terminals can be used to ensure a secure connection. There are five I2C interfaces, four customizable through-holes and one 3.5mm jack connection. There is one UART interface and one SPI interface. When the SPI interface is not in use, three digital input pins are available. D5, D6, D7. One analog pin is also available. ESP8266 was chosen as a board for its high clock speed, flash memory, and wifi connection, while having a very cheap cost (~\$2). There are designated spots for pull-up resistors, as well as customizable voltage options. This board was manufactured and used in all testing.

Printed Circuit Board (PCB)

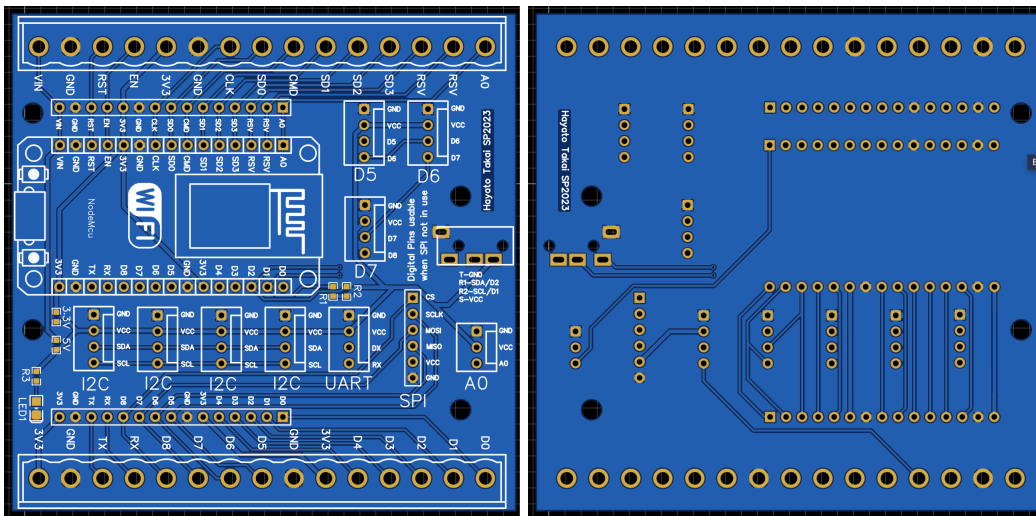


Fig. 5 a,b: Front (Left), Back (Right) of PCB

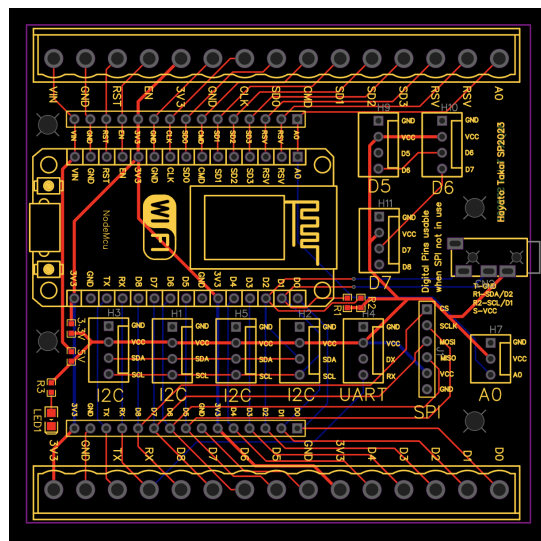
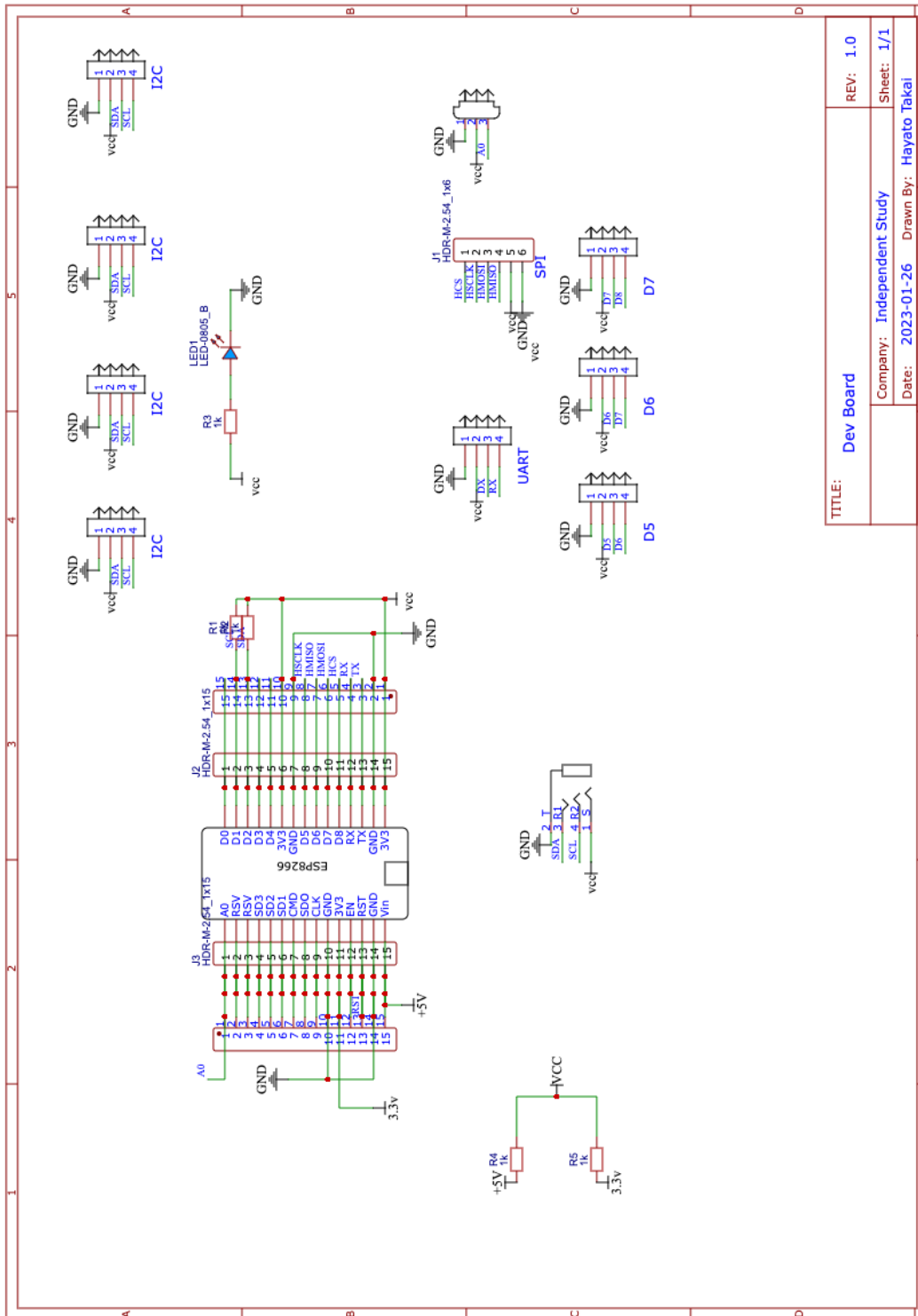


Fig. 6: Wire Traces for Development Board

Schematic



TITLE: Dev Board	REV: 1.0
Company: Independent Study	Sheet: 1/1
Date: 2023-01-26	Drawn By: Hayato Takai

Fig. 7: Schematic for Development Board

Board Comparison

	ESP8266 (NodeMCU)	ESP32 (Devkit)	Arduino Nano (Atmega328P)
Flash	4mb	16mb	32Kb
Clock Speed	160MHz	160MHz	16MHz
Output Voltage (DC)	3.3V	3.3V	5V DC
Input Voltage (DC)	5V - 12V	5V - 12V	Mac 20V
Low Power	20 μ A	10 μ A	6mA
Digital Pins	17	6	14
Analog Pins	1	8	6
PWM	4	16	6
I2C	2	2	1
UART	2	2	1
SPI	2	4	1
WiFi	Yes	yes	No
Bluetooth	No	yes	No
Cost	\$3-\$6	\$6-\$12	\$10-\$20
Datasheet	https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf	https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf	https://docs.arduino.cc/resources/datasheets/A00066-datasheet.pdf

Warning, that when using ESP8266, the GPIO has different pinout numbers. See image below

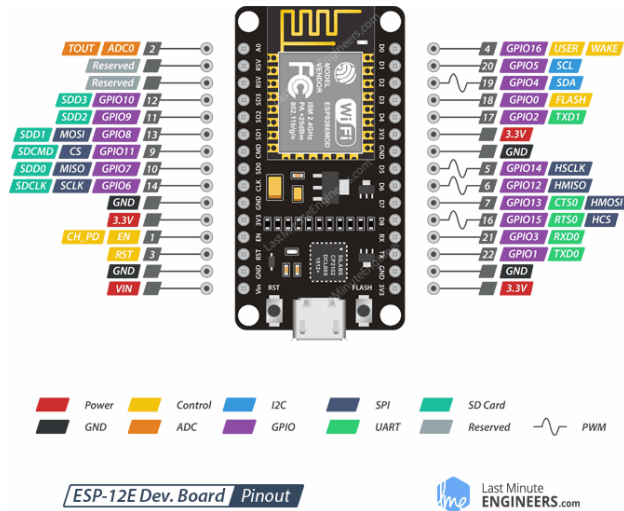


Fig. 8: Image from <https://coeleveld.com/esp8266-esp-12e-big-digits-ntp-clock-with-date-time-temperature-humidity-pressure-barometer/>

Skills

Connections

Soldering: <https://www.youtube.com/OKbJxytERvg>

Solid core vs stranded wire: <https://www.youtube.com/UcZX-s3lEN4>

Logic Level Shifting

Logic level shifting is a way to make the +5V output of a microcontroller such as an Arduino Uno, compatible with +3.3V logic devices (Such as I2C devices without voltage controllers).

<https://www.youtube.com/Ux0WYpDA>

General:

Personal favorite youtube channel for electronics (Playlist:

<https://www.youtube.com/KoSatCnclhE>

Data Transfer

Data transfer can be done by connecting to a mobile device via bluetooth or via pogo pins

Bluetooth

The easiest way to use bluetooth will be to read data off of an SD card in block or “chunks” and transmit each block at a time over a serial bluetooth connection to a mobile device. Since ESP8266 does not support bluetooth, recommended boards are ESP32 or nrf52840 or stm32wb. See “Bluetooth Serial Transmit from Sd Card” in the code section below.

An mobile app for receiving serial data is required.

https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal&hl=en&pli=1

Warning: Due to the limited bluetooth apps available, there is a high chance that apple does not allow SPP (Serial Port Profile) or “Classic Bluetooth” devices and interfaces.

A workaround is to use BLE, which is a bit more complicated, and data logging files may not be suitable.

IOS: <https://apps.apple.com/us/app/nrf-connect/id1054362403>

Pogo Pins:

What are pogo pins: https://youtu.be/NGZ_vd6qmeQ

One method of transferring data is having 4 waterproof contacts that are pressed on by a I2C SD card. When the device detects that the new SD card is connected, it transfers data from the internal SD card to the external SD card, using the same block by block method.

<https://www.digikey.com/en/products/detail/mill-max-manufacturing-corp/878-20-004-00-01100/16630537>

Deep Sleep

Deep Sleep is a state where a microcontroller shuts off everything high frequency and only leaves the clock and wake circuitry. Power consumption values range wildly since there are so many factors that draw power. Development boards will use more power in deep sleep than their chip alone. In usage, more power will be used than written deep sleep power consumption values as energy is used to wake the device and perform a task.

In an idle state, an ESP32 can use 15 μ A to 400mA. Assuming Wifi is on, the ESP32 dev board consumes typically around 58mA at 3.3V in idle.

$$\text{Watt's Law: } W = V * A * p.f = 3.3 * 58\text{mA} = 191\text{mW}$$

$$\text{Power Consumption per day: } 4.6\text{Wh}$$

On a 5000mAh battery, $5000\text{mAh} / 58\text{mA} = 86$ hours

However, batteries usually drop in voltage before they deplete so this is most likely lower.

In deep sleep, power consumption usually drops to around 4.1mA. The power consumption is a lot higher than what the data sheet states due to power loss from the voltage regulator and CP2102N chip.

$$\text{Watt's Law: } W = V * A * p.f = 3.3 * 4.1\text{mA} = 13.5\text{mW}$$

On a 5000mAh battery, $5000\text{mAh} / 4.1\text{mA} = 1220$ hours

This is a lot longer but since the device will need to wake up from time to time to record data, this value will be lower.

This can be done with:

```
ESP.deepSleep(uint32 time_in_us);
```

Prototyping

CAD (Fusion 360)

Can probably reuse the tutorials from last year as it still fits the curriculum

Laser Cutting

What is laser cutting?

- <https://www.twi-global.com/technical-knowledge/faqs/what-is-laser-cutting>

CO2 laser cutting primer

- <https://aplazer.com/how-it-works-co2-laser-cutters/>

3D printing

Option to reuse old 3D printing primers but I have put a few more below

- What is it?
 - <https://3dprinting.com/what-is-3d-printing/>
- Types of 3D printing
 - <https://ultimaker.com/learn/what-is-3d-printing>
- What 3D printing made possible
 - <https://all3dp.com/2/3d-printing-innovations-innovative-3d-printing/>
- Tolerances
 - <https://all3dp.com/2/3d-printing-tolerances-test-fdm/>

Hot glue

Makerspace hot glue tutorial

Prototyping

Engineering design process primer

Weather Station Design:

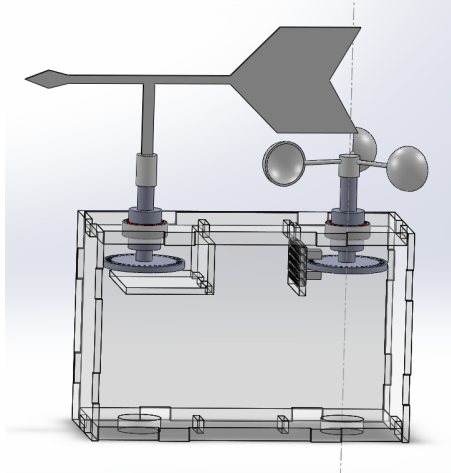
- Example of a hobbyist kit
 - https://www.sparkfun.com/products/15901?gclid=CjwKCAjwrpOiBhBVEiwA_473dPWXGvMr1Ggk4m2xneXzYnAoSZ5ZAXWTkg5ffRxxC4wBg906Btw6BxoC2icQAvD_BwE
 - Cons
 - Only measures 8 directions (not 360 degrees)
 - Uses drag and drop coding (not open source and very limited)
- Other examples
 - Upgraded sparkfun weather station
 - <https://www.instructables.com/Web-Connected-Weather-Station/>
 - DIY weather station
 - <https://www.instructables.com/DIY-Standalone-Weather-Station-Powered-by-Arduino/>
 - IoT (long range, low power) weather station
 - <https://www.instructables.com/Esay-IoT-Weather-Station-With-Multiple-Sensors/>

Weather Station Design:

- Constraints
 - Has to be weather resistant
 - Can't be too bulky
 - Should be inexpensive
- Optimization
 - Should be able to be put in a remote area
 - Should last awhile
 - Data should be relatively precise

Thought process:

- Could be helpful to put the anemometer and wind vane next to each other to save space instead of making them into one unit
 - Could have a shaft of different length so one fits on top of the other (see picture)



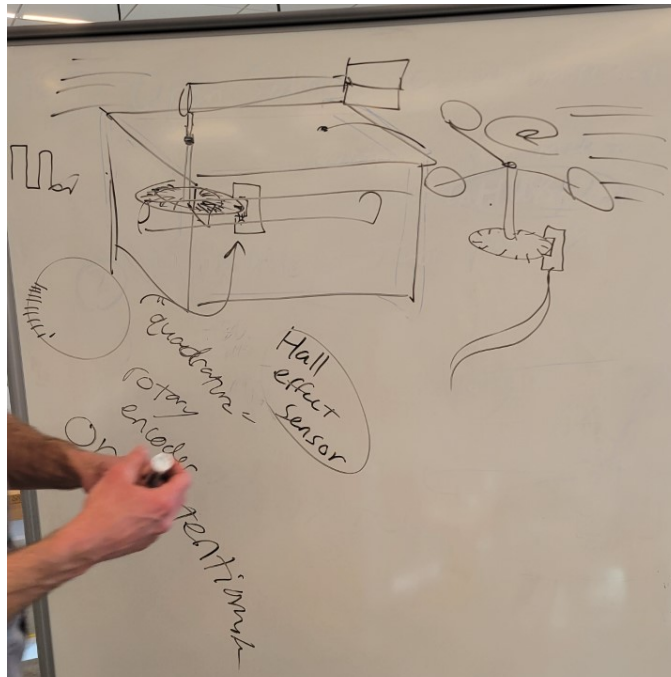
- Would be nice to reuse parts to save from having all unique pieces
- Function Tree:



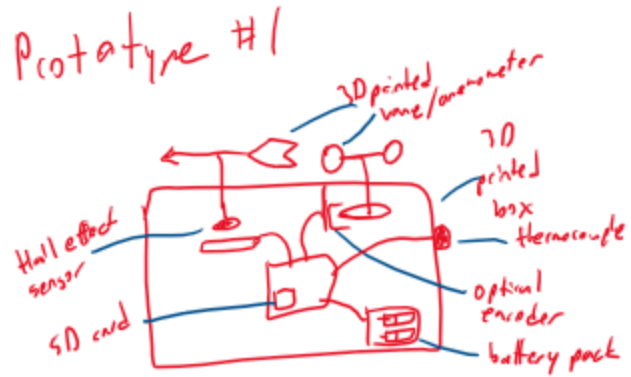
Morph Chart:

stay outside	box 	sphere 	cover 
store data	SD 	USB 	computer 
measure temperature	thermistor 	thermo couple 	X
measure wind speed	optical encoder 	motor voltage 	Velocity formula D/T 
wind direction	hall effect sensor 	optical encoder 	limit switch 
Last 1 month	battery + no wifi  + 	USB connect 	solar 

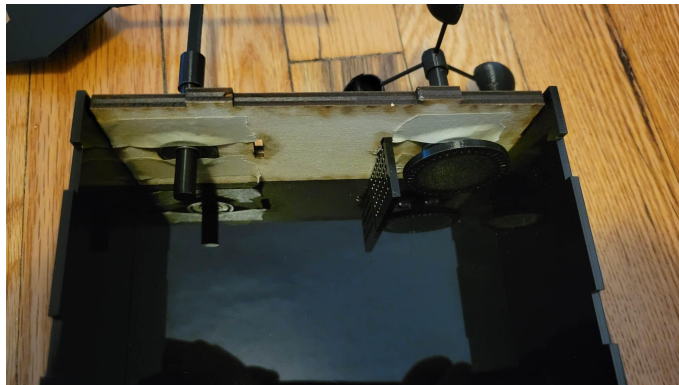
Potter's Prototype:



Prototype Design:



- Part selection:
 - Shaft should be 3D printed (it is round so would be hard to laser cut)
 - Box should be laser cut (more straight edges and acrylic might be more weather proof)
 - Wind vane and anemometer should be 3d printed (usually are plastic parts and unique shapes)
- Prototype pictures



○



○

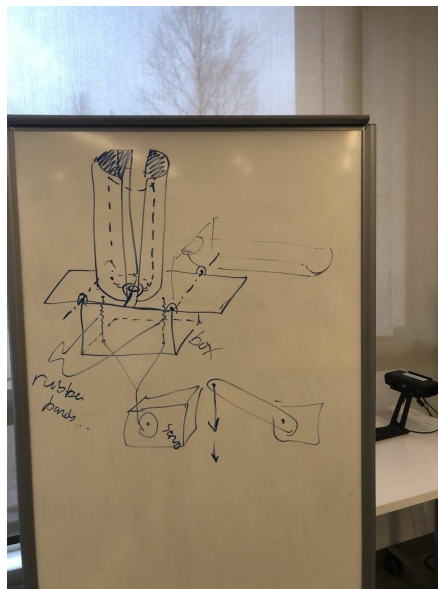
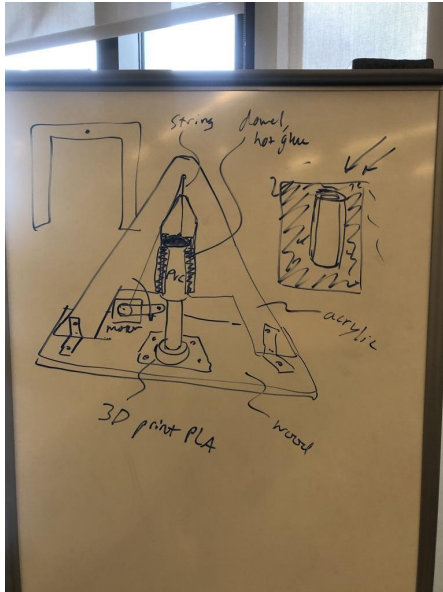
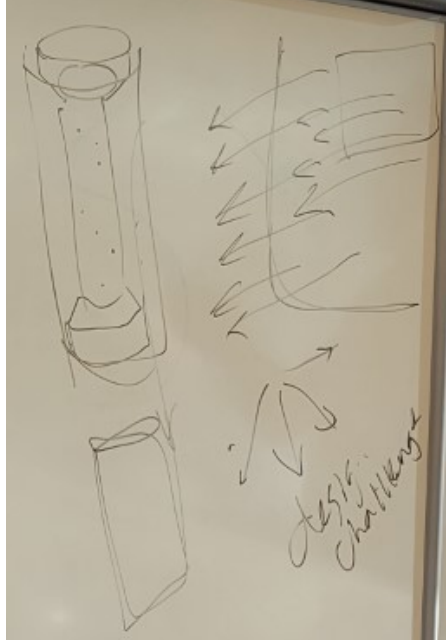


-
- Issues with prototype
 - Tolerance errors with 3D printed parts +/- 0.25mm
 - Shaft collar thickness (bolt dragging along bearing)
 - Bearing friction (doesn't spin well)
- Improvement recommendations
 - Find a more frictionless way to move pieces (capture small wind direction changes)
 - Refine the tolerances for laser cuts and 3D prints (weatherproofing)
 - Add a hinge design for easy access

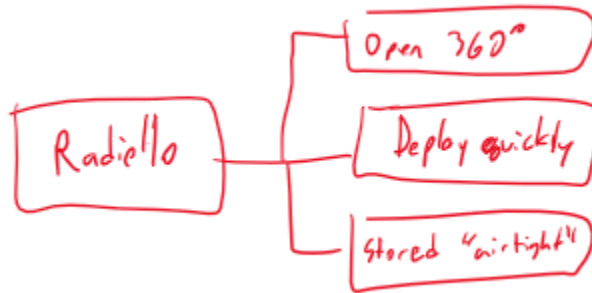
Radiello Design:

- Introduction (<https://radiello.com/>)
 - Radiello is an air quality monitoring system that detects specific particles in the air like pollen, and then changes color accordingly (indicating that specific particles are present in the air). Radiello was designed with simplicity in mind, as it is electricity free, lightweight, easy-to-produce, and has multiple uses.
 - The issue with Radiello is sometimes there are false positives and also transport through places that don't need to be measured. To combat this, students are tasked with designing a contraption to keep the Radiello away from detecting when not needed, and then release it quickly through a contraption
- Constraints
 - Radiello must have 360 degree freedom when extended
 - Radiello should be relatively hidden away when stored (doesn't have to be fully airtight)
- Optimization
 - As light as possible
 - As small as possible

Dr. Potter's idea:



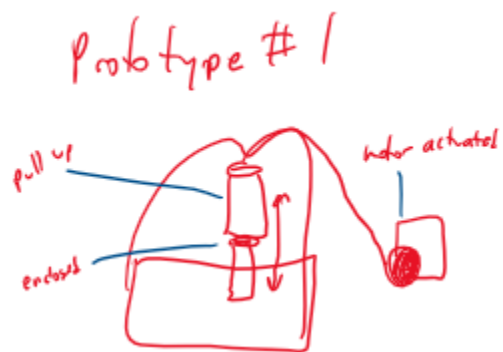
Function Tree:



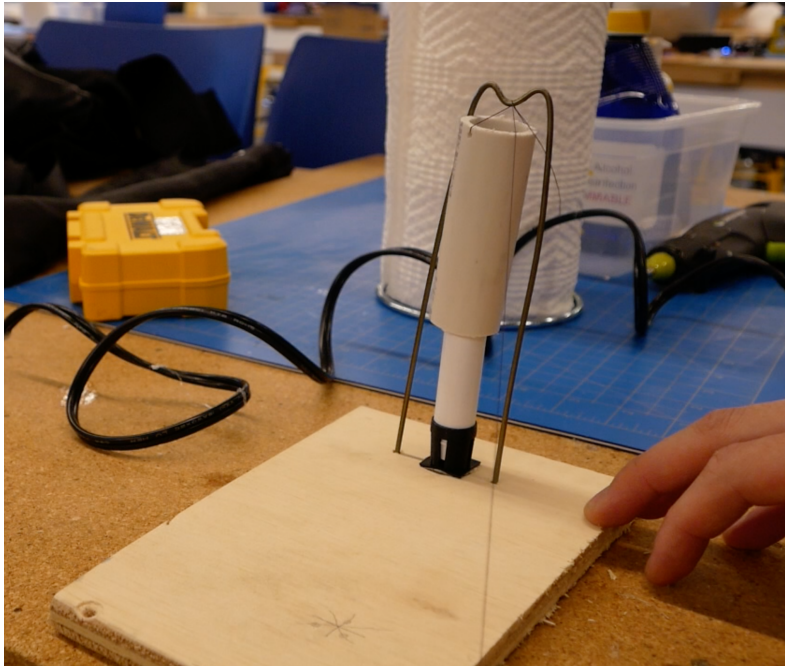
Morph Chart:

Open $> 60^\circ$	push up 	flip out 	push out 
deploy quickly	motorize 	spring loaded 	actuator/lever 
'air tight'	enclosed 	cap 	doors 

Prototype Design:



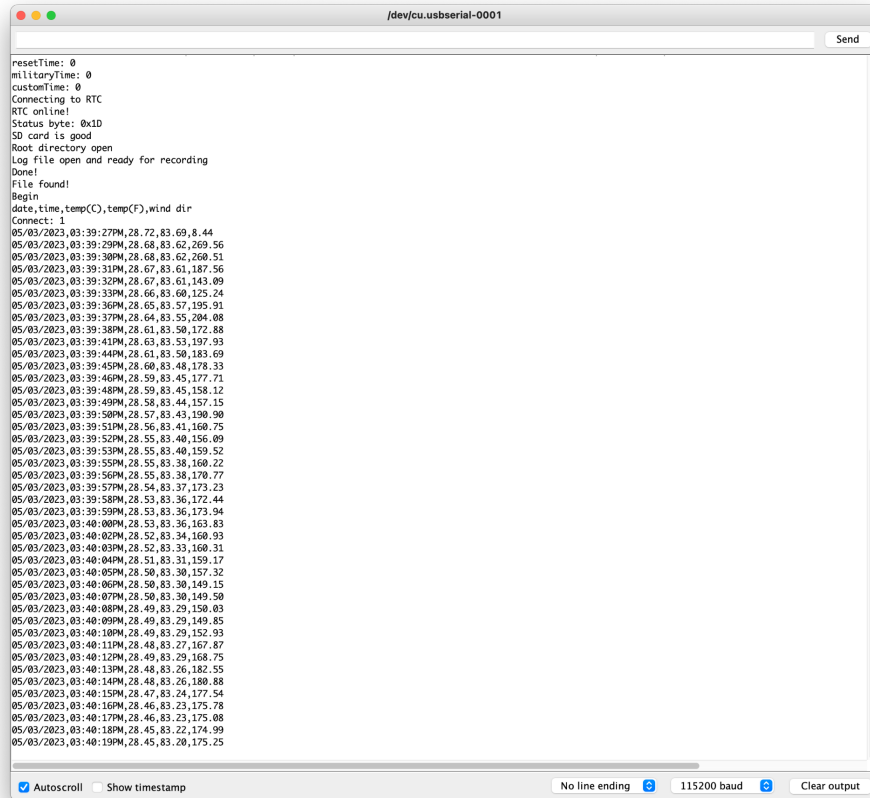
Radiello Cover Design



The radiello design was built to have students get hands on experience with building. The design uses wire from a hanger that was cut and bent. Two holes were drilled into a piece of plywood to secure the wire in place. A length of pvc pipe was cut and two holes were tapped and drilled in a vice to pass through a string. This string will be attached to the arm of a servo motor that lifts and lowers the pvc. There is a 3D printed piece attached to the top of the radiello that creates an airtight-like seal that also acts as a track for the pvc pipe to slide along.

Code

I2C Data Logging Sample Code



```
resetTime: 0
militaryTime: 0
customTime: 0
Connecting to RTC
RTC online!
Status byte: 0x1D
SD card is good
Root directory open
Log file open and ready for recording
Done!
File found!
Begin
date,time,temp(C),temp(F),wind_dir
connect: 1
05/03/2023,03:39:27PM,28.72,83.69,8.44
05/03/2023,03:39:29PM,28.68,83.62,269.56
05/03/2023,03:39:30PM,28.68,83.62,260.51
05/03/2023,03:39:31PM,28.67,83.61,187.56
05/03/2023,03:39:32PM,28.67,83.61,143.09
05/03/2023,03:39:33PM,28.66,83.60,125.24
05/03/2023,03:39:36PM,28.05,83.57,195.91
05/03/2023,03:39:37PM,28.64,83.55,204.08
05/03/2023,03:39:38PM,28.61,83.50,172.88
05/03/2023,03:39:41PM,28.63,83.53,197.93
05/03/2023,03:39:44PM,28.61,83.50,183.69
05/03/2023,03:39:45PM,28.60,83.48,178.33
05/03/2023,03:39:46PM,28.59,83.45,177.71
05/03/2023,03:39:48PM,28.59,83.45,158.12
05/03/2023,03:39:49PM,28.58,83.44,157.15
05/03/2023,03:39:50PM,28.57,83.45,190.90
05/03/2023,03:39:51PM,28.56,83.41,160.75
05/03/2023,03:39:52PM,28.55,83.40,156.09
05/03/2023,03:39:53PM,28.55,83.40,159.52
05/03/2023,03:39:55PM,28.55,83.38,160.22
05/03/2023,03:39:56PM,28.55,83.38,170.77
05/03/2023,03:39:57PM,28.54,83.37,173.23
05/03/2023,03:39:58PM,28.53,83.36,172.44
05/03/2023,03:39:59PM,28.53,83.36,173.94
05/03/2023,03:40:00PM,28.53,83.36,163.83
05/03/2023,03:40:02PM,28.52,83.34,160.93
05/03/2023,03:40:03PM,28.52,83.33,160.31
05/03/2023,03:40:04PM,28.51,83.31,159.17
05/03/2023,03:40:05PM,28.50,83.30,157.32
05/03/2023,03:40:06PM,28.50,83.30,149.15
05/03/2023,03:40:07PM,28.50,83.30,149.50
05/03/2023,03:40:08PM,28.49,83.29,150.03
05/03/2023,03:40:09PM,28.49,83.29,149.85
05/03/2023,03:40:10PM,28.49,83.29,152.93
05/03/2023,03:40:11PM,28.48,83.27,167.87
05/03/2023,03:40:12PM,28.49,83.29,168.75
05/03/2023,03:40:13PM,28.48,83.26,182.55
05/03/2023,03:40:14PM,28.48,83.26,180.88
05/03/2023,03:40:15PM,28.47,83.24,177.54
05/03/2023,03:40:16PM,28.46,83.23,175.78
05/03/2023,03:40:17PM,28.46,83.23,175.08
05/03/2023,03:40:18PM,28.45,83.22,174.99
05/03/2023,03:40:19PM,28.45,83.20,175.25
```

Sample Output (exported as csv)

//Author: Hayato Takai 5/3/2023

/*Description:

This code was written for a datalogging weather station for Independent Study SP2022.

Hardware requirements:

- Sparkfun Qwiic RV8803 RTC
- Sparkfun Openlog DEV15164
- Sparkfun TMP117 Temperature
- AS5600-SO_EK_AB AMS OSRAM Magnetic Encoder

*/

```

//
*****Settings*****
*****
****
boolean resetTime = false; // if true, resets RTC time
boolean militaryTime = false; // if true, uses 24hr clock
boolean customTime = false; // if true, sets the RTC to custom time
String myFile = "test.txt";
// *****Hardware
Connections*****
*****
// Plug the RTC into the Qwiic port on your microcontroller or on your Qwiic shield/adapter.
// If you are using an adapter cable, here is the wire color scheme:
// Black=GND, Red=3.3V, Blue=SDA, Yellow=SCL
// Open the serial monitor at 115200 baud

//
*****Libraries*****
*****
****
#include <SparkFun_RV8803.h> //Get the library here:http://librarymanager/All#SparkFun_RV-8803
RV8803 rtc; //Create RTC instance

#include "SparkFun_Qwiic_OpenLog_Arduino_Library.h"
OpenLog myLog; //Create Logger instance
int ledPin = LED_BUILTIN; //Status LED connected to digital pin 13

#define STATUS_SD_INIT_GOOD 0
#define STATUS_LAST_COMMAND_SUCCESS 1
#define STATUS_LAST_COMMAND_KNOWN 2
#define STATUS_FILE_OPEN 3
#define STATUS_IN_ROOT_DIRECTORY 4

// The default address of the device is 0x48 = (GND)
#include <SparkFun_TMP117.h> // Used to send and receive specific information from our sensor
TMP117 sensor; // Initalize sensor

#include "AS5600.h"
AS5600 as5600; // use default Wire
// *****Custom
Time*****
*****
// The below variables control what the date and time will be set to
int sec = 2;

```

```

int minute = 47;
int hour = 14; //Set things in 24 hour mode
int date = 2;
int month = 3;
int year = 2020;
int weekday = 2;

// *****Set
Up*****
*****

void setup(){
  pinMode(ledPin, OUTPUT);
  Wire.begin();    // begin I2C communication
  // Wire.setClock(400000); //Go super fast
  Serial.begin(115200); // serial baud rate
  Serial.println("Setting: ");
  Serial.println("resetTime: " + String(resetTime));
  Serial.println("militaryTime: " + String(militaryTime));
  Serial.println("customTime: " + String(customTime));
  // *****RTC Set
Up*****
*****

  Serial.println("Connecting to RTC");
  if (rtc.begin() == false){
    Serial.println("Device not found. Please check the wiring. Freezing.");
    while (1);
  }
  Serial.println("RTC online!");
  // Use the time from the Arduino compiler (build time) to set the RTC
  // Keep in mind that Arduino does not get the new compiler time every time it compiles.
  // To ensure the proper time is loaded, open up a fresh version of the IDE and load the sketch.
  // Also note that due to upload times, compiler time may be a little bit off on seconds/hundredths
  if (resetTime) {
    Serial.println("Resetting Time");
    if (customTime) {
      Serial.println("Setting device to Custom Time");
      if (rtc.setTime(sec, minute, hour, weekday, date, month, year) == false) {
        Serial.println("Custom time set: Fail");
      } else {
        Serial.println("Custom time set: Success");
      }
    } else {
      Serial.println("Setting device to Compiler Time");
      if (rtc.setToCompilerTime() == false) {

```

```

    Serial.println("Custom time set: Fail");
  } else {
    Serial.println("Compiler time set: Success");
  }
}
if (militaryTime) {
  Serial.println("Using 24hr clock");
  rtc.set24Hour();
} else {
  Serial.println("Using 12hr clock");
}
}
// *****Openlog Set
Up*****
*****
myLog.begin();
byte status = myLog.getStatus();
Serial.print("Status byte: 0x");
if (status < 0x10) Serial.print("0");
Serial.println(status, HEX);
if (status == 0xFF){
  Serial.println("OpenLog failed to respond. Freezing.");
  while (1); //Freeze!
}
if (status & 1 << STATUS_SD_INIT_GOOD){
  Serial.println("SD card is good");
}
else{
  Serial.println("SD init failure. Is the SD card present? Formatted?");
}
if (status & 1 << STATUS_IN_ROOT_DIRECTORY){
  Serial.println("Root directory open");
}
else{
  Serial.println("Root failed to open. Is SD card present? Formatted?");
}
if (status & 1 << STATUS_FILE_OPEN){
  Serial.println("Log file open and ready for recording");
}
else{
  Serial.println("No log file open. Use append command to start a new log.");
}
Serial.println("Done!");

```

```

long sizeOfFile = myLog.size(myFile);
if (sizeOfFile == -1){ //File does not exist. Create it.
  Serial.println(F("File not found, creating one..."));
  myLog.append(myFile); //Create file and begin writing to it
  myLog.syncFile();
} else {
  Serial.println("File found!");
  myLog.append(myFile); //Create file and begin writing to it
}
// *****TMP117 Set
Up*****
*****

if (sensor.begin() == true){ // Function to check if the sensor will correctly self-identify with the proper
Device ID/Address
  Serial.println("Begin");
} else {
  Serial.println("Device failed to setup- Freezing code.");
  while (1); // Runs forever
}
Serial.println("date,time,temp(C),temp(F),wind dir");
myLog.println("date,time,temp(C),temp(F),wind dir");

// *****AS5600 Set
Up*****
*****

as5600.begin(4); // set direction pin.
as5600.setDirection(AS5600_CLOCK_WISE); // default, just be explicit.
int b = as5600.isConnected();
Serial.print("Connect: ");
Serial.println(b);}
//
*****Loop*****
*****
***

void loop(){
  float dir=as5600.rawAngle() * AS5600_RAW_TO_DEGREES;
  if (sensor.dataReady() == true){ // Function to make sure that there is data ready to be printed, only
prints temperature values when data is ready
  if (rtc.updateTime() == true){ // Updates the time variables from RTC
    myLog.append(myFile); //Create file and begin writing to it
    String currentDate = rtc.stringDateUSA(); // Get the current date in mm/dd/yyyy format (we're weird)
    String currentTime = rtc.stringTime(); // Get the time
    Serial.print(currentDate);
    myLog.print(currentDate);

```



```
Serial.print(",");
myLog.print(",");
Serial.print(currentTime);
myLog.print(currentTime);
Serial.print(",");
myLog.print(",");
float tempC = sensor.readTempC();
float tempF = sensor.readTempF();
// Print temperature in °C and °F
Serial.print(tempC);
myLog.print(tempC);
Serial.print(",");
myLog.print(",");
Serial.print(tempF);
myLog.print(tempF);
Serial.print(",");
myLog.print(",");
Serial.print(dir);
myLog.print(dir);
} else {
  Serial.print("RTC read failed");
}
Serial.println("");
myLog.println("");
myLog.syncFile();
}
}
```

Bluetooth Serial Transmit from Sd Card

//Author: Hayato Takai 5/3/2023

/*Description:

This code was written for reading data off of an sd card and transmitting the data over bluetooth for Independent Study SP2022.

Hardware requirements:

- Serial Bluetooth App

https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal&hl=en&pli=1

- ESP32

*/

//

```
*****Libraries*****
*****
*****
```

#include

"BluetoothSerial.h"//<https://github.com/espressif/arduino-esp32/tree/master/libraries/BluetoothSerial>

#include "SparkFun_Qwiic_OpenLog_Arduino_Library.h"

OpenLog myLog; //Create Logger instance

// Check if Bluetooth configs are enabled

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)

#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it

#endif

// Bluetooth Serial object

BluetoothSerial SerialBT;

String myFile = "test.txt";

void setup() {

 Serial.begin(115200);

 SerialBT.begin("ESP32");

}

// *****Set

Up*****

void loop() {

 long sizeOfFile = myLog.size(myFile);

 if (sizeOfFile > -1){

 Serial.print(F("Size of file: "));

 Serial.println(sizeOfFile);

 //Read the contents of myFile by passing a buffer into .read()

```

//Then printing the contents of that buffer
byte myBufferSize = 200;
byte myBuffer[myBufferSize];
//myLog.read(myBuffer, myBufferSize, myFile, 4); //Doesn't yet work
myLog.read(myBuffer, myBufferSize, myFile); //Load myBuffer with contents of myFile

//Print the contents of the buffer
Serial.println("File contents:");
for (int x = 0 ; x < myBufferSize ; x++){
  SerialBT.print(myBuffer[x]);
  Serial.write(myBuffer[x]);
}
Serial.println("\nDone with file contents");
} else{
  Serial.println(F("Size error: File not found"));
}

Serial.println(F("Done!"));
myLog.syncFile();
}

```