Kairos! Portable AQI Sensor (EE 156)

Note: Kairos is my second attempt at this project. My initial design, named Stellaris, was intended to serve as an embedded solar MPPT charge controller for portable device charging. It was my first time designing a circuit board, and I encountered several challenges, particularly in understanding what was technically feasible. These challenges led to a number of oversights that ultimately rendered the fabricated board unusable. Some key issues included:

1. Lack of access to the SAMD51's D+/D- pins, as I didn't connect any traces to the microcontroller's physical pins, making it impossible to use CircuitPython for programming.
2. Power stability problems due to the absence (or poor placement) of EMI capacitors, causing the 3.3V rail to be highly unstable. The custom MPPT charge controller also had significant design flaws.
3. Choosing sensors without existing CircuitPython libraries, which would have required extensive development in C to make the system functional.

Ultimately, I decided to pivot from Stellaris to Kairos, as I felt it offered a more valuable learning experience for my first microcontroller PCB project. In hindsight, I'm glad I made the switch, even though it meant reworking some of the original functionality.

Project Overview

The objective of this project was to develop a low-cost, portable AQI (Air Quality Index) monitor that logs key air quality metrics and stores them for long-term tracking.

This is driven by the worsening air quality due to climate change, particularly in regions like California where wildfires contribute to increased particulate matter in the atmosphere. Affordable sensors, such as Kairos, offer a scalable solution for monitoring air quality across large areas, providing national agencies with valuable data to analyze climate patterns, identify hotspots, and allocate resources more effectively.

A key advantage of Kairos is its portability and accessibility. Designed to be carried by individuals, it requires minimal infrastructure, reducing the overhead costs typically associated with large-scale environmental monitoring. This enables on-the-ground data collection and empowers individuals to better understand and respond to the air quality and environmental hazards in their immediate surroundings, while also meaningfully contributing to a larger, community or state-wide, AQI dataset. This version of Kairos serves as a proof of concept and has the following features:

1. Embedded compute
   a. The **ATMEL SAMD51J19**, an accessible chip, with 120MHz CPU, 256 KB RAM, and plenty of IO buses to handle sensor data and logging.
2. Instrumentation
   a. A **Bosch SensorTech BME 688** — enabling high-accuracy pressure, humidity, temperature, Volatile Organic Compound (VOC) and Volatile Sulfur Compound (VSC) monitoring.
   b. An **LTR-329ALS-01** — enabling visible- and IR-range light intensity measurements.
3. Communication modalities
   a. A **USB bus** — for communication with a host computer
   b. A **microSD** bus — for time-stamped ***logging*** of sensor values. Using an external SD card allows for prolonged operation without needing to pause logging
   c. A **display** — for on the go monitoring of sensor values

4. Two different power modes
    a. **USB Power** — embedded voltage regulators enable Kairos to be used in a stationary setting, say a roof of a building, for extended periods of time
    b. **Battery Power** — a 2500mAh LiPO and embedded charging circuitry allow Kairos to be taken on the go, for better-monitoring frequently travelled environments without requiring frequent charging.

While this version of Kairos demonstrates proof of concept, future iterations should expand its capabilities with the following features:

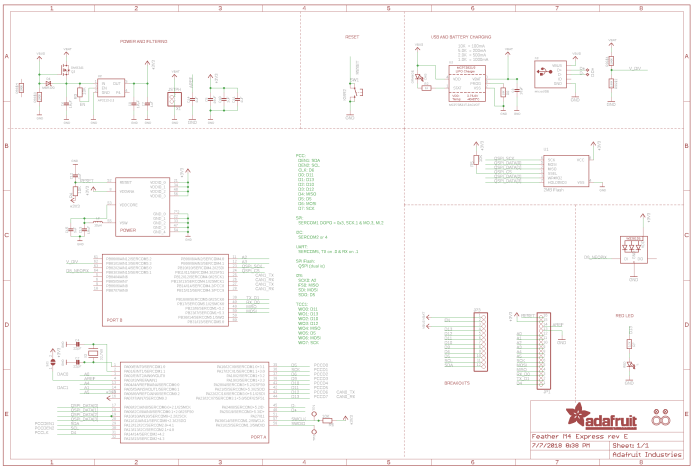
1. **UV Sensor**: To capture sun intensity across more (and more dangerous) wavelengths, providing a more comprehensive solar exposure assessment.
2. **GPS Module**: To enable precise tracking of Kairos' location, facilitating the creation of accurate heatmaps for key AQI metrics.
3. **Bluetooth Syncing**: To allow seamless data transfer to mobile phones, removing the need to manually retrieve data from an SD card.
4. **A Centralized Server**: To aggregate data from multiple Kairos units, generating regional-level heatmaps and providing actionable insights from averaged data points across different areas.
5. **A rugged, waterproof, miniaturized shell for on the go use.**

Design

This PCB is heavily inspired by Adafruit's Feather M4 Express. See below:

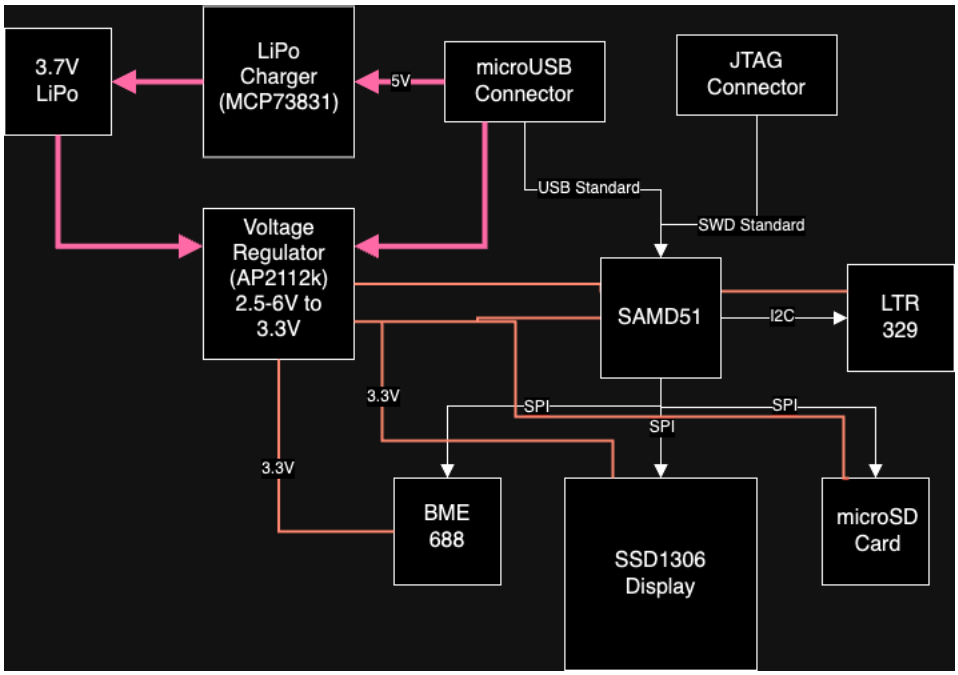
Adafruit Learning System

https://learn.adafruit.com/assets/57242



The guiding principle here being *not* trying to reinvent the wheel

System block diagram



Estimated power consumption

Component	Peak Current Draw	Voltage	Peak Power
LTR329	220uA	3.3V	726uW
BME688	3.9mA	3.3V	12.87mW
SAMD51	130uA (assume 2MHz max for SPI/I2C duplexing)	3.3V	0.429mW
SSD1306 (Display)	15mA	3.3V	49.5mW

Total peak power: 63.525mW

Total peak current: 19.25mA

Battery used is 3.7V, 2500mAh LiPo. Let's assume voltage regulator/overall power IC efficiency is 60%.

Then the total power drawn from the battery is:

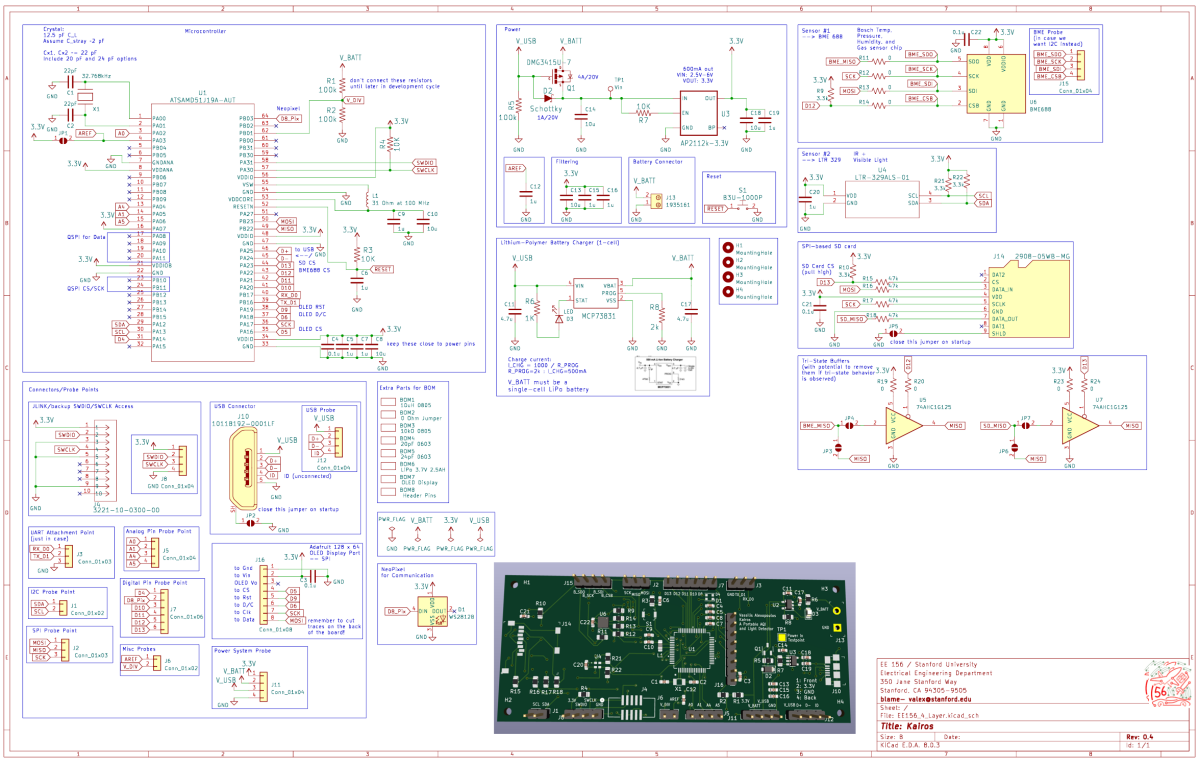
$$P_{PowerIC} = \frac{63.525mW}{0.6} = 105.875mW$$

Which leads to a current draw of:

$$I_{\text{total}} = \frac{P_{PowerIC}}{V_{\text{battery}}} = \frac{105.875\,\text{mW}}{4.2\,\text{V}} = 25.21\,\text{mA}$$

Given a 2500mAh capacity, this yields and **estimated runtime of ~100hrs**

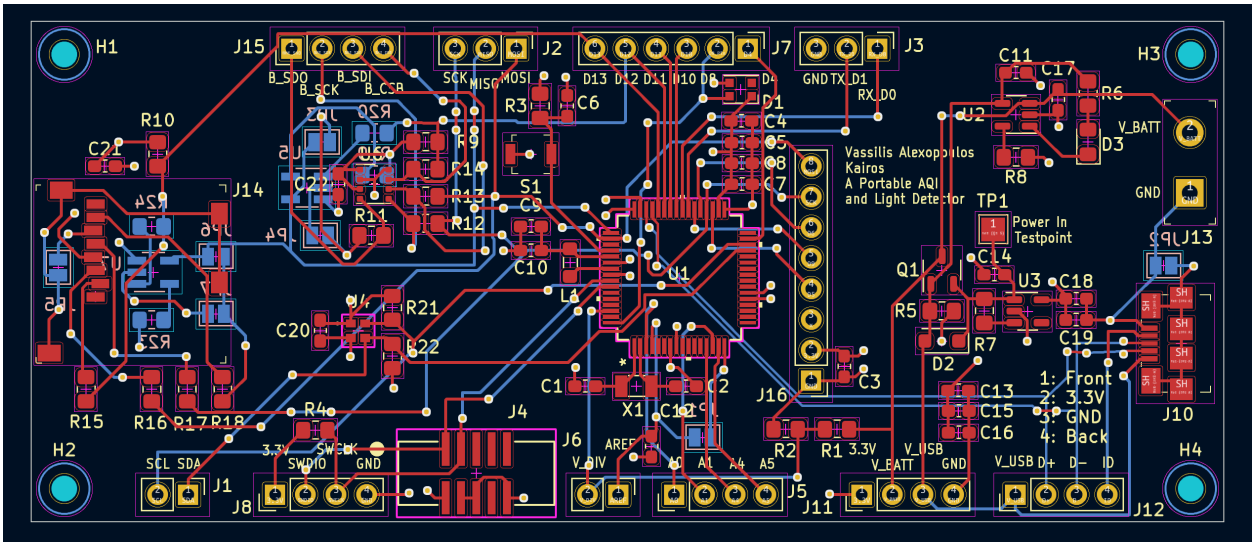
Schematic

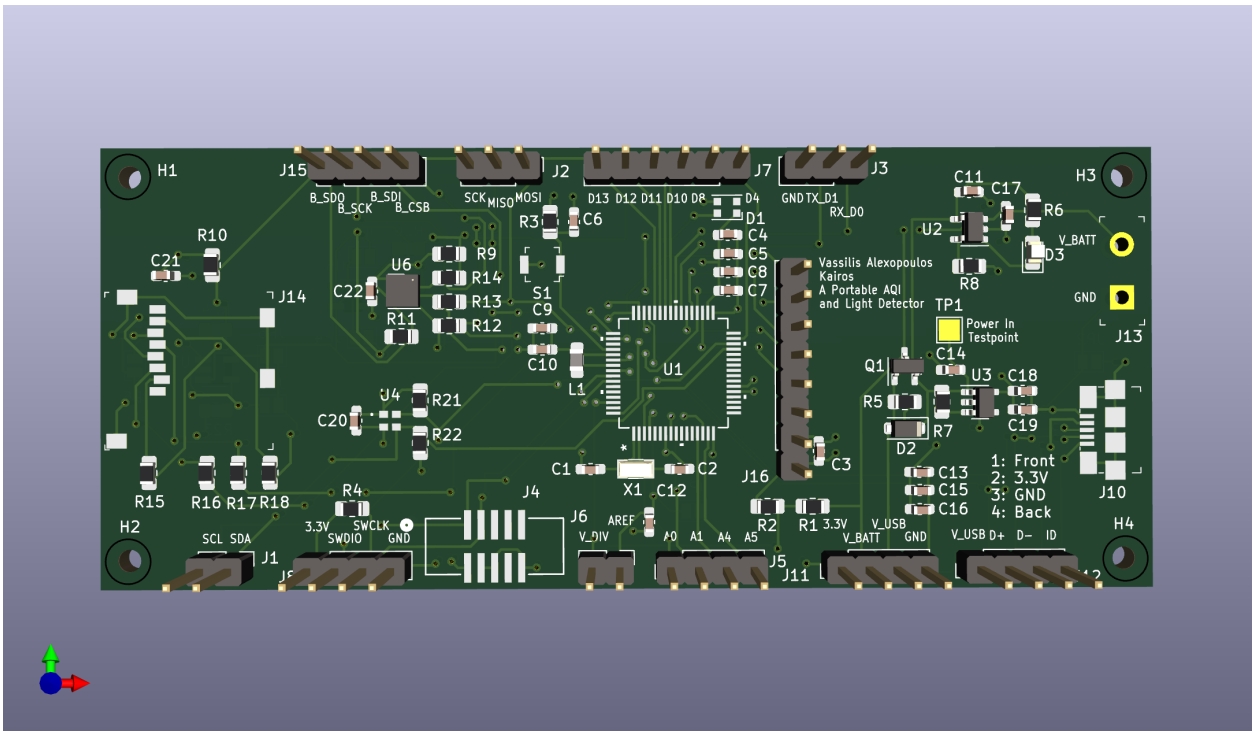


Generally, I designed Kairos to be as robust as possible. I included backup access points for probing critical lines, jumpers for easy reconfiguration of circuit sections, and “just in case” tri-state buffers to prevent potential SPI bus conflicts between the SD card and BME688.

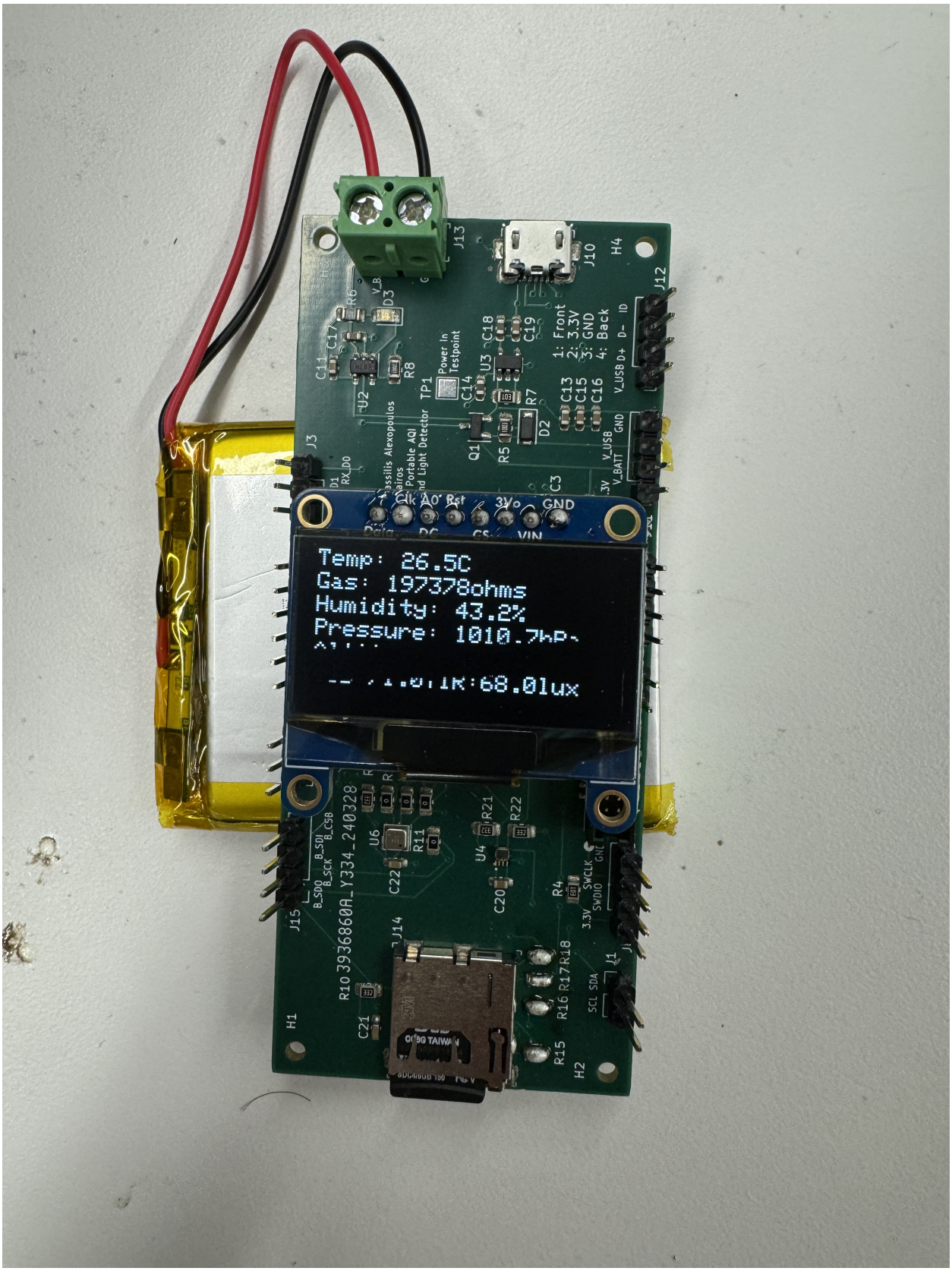
Spending extra time during the schematic design to consider how I would handle board bring-up and potential failures made the entire process **significantly smoother,** even if it meant using extra components and board space that ultimately weren’t strictly necessary.

Layout





Layout prioritized the accessibility of headerpins while keeping the SAMD51 as central as possible. The vertical row of headerpins in the middle of the board allows direct mounting of the display to form a single assembly. See below:


Temp: 26.5C
Gas: 197378ohms
Humidity: 43.2%
Pressure: 1010.7hP
68.0lux
Clk A0 Rst 3Vo GND
DC CS VIN
Power In Testpoint
Portable AQI
Light Detector
1: Front
2: 3.3V
3: GND
4: Back
V_USB D+ D- ID
V_USB GND
V_BATT
3V
RX_D0
D1
B_SDO B_SCK B_SDI B_CS8
3.3V SWDIO SWCLK GND
SCL SDA
3936860A_Y334_240328

Bringup

Note: since I'm not doing anything that requires highly accurate timing, I'm opting to use the SAMD51's internal crystal oscillator. YMMV.

Testing Process

1. Visually inspect the bare board
   - Check for mechanical damage (burrs, scratches, gouges)
   - Examine manufacturing issues like copper bridging or pinholes
2. Check for shorts
   - Use a multimeter to test for shorts and ensure proper continuity on power rails and IC inputs
4. Unit test on board power regulators
   - Place power systems
   - Apply test voltage, verify that 5V and 3.3V power rails are stable
   - Check for temperature anomalies
   - Confirm battery charge IC can appropriately charge and discharge battery
     - Use a power resistor to bring LiPo to low state of charge, then charge via IC and USB power
     - Remove USB power and verify 3.3V rail is stable
5. Test microcontroller functionality
   - Reflow the microcontroller and the power systems onto the board, conduct a visual inspection for soldering issues
   - Check shorts between microcontroller legs with a multimeter
   - Verify that JTAG is connected properly, inspect JTAG pins
   - Use a **Segger JLink** and **JLINK lite** to flash the microcontroller (**IT'S ALIVE** 🦇)
     - 4000 baud, start at the base memory address
     - *I used JLINK Lite on it's own to flash the bootloader, didn't bother with Atmel Studio*
       - Bootloader used: **j19a_no_crystal.bin**
     - Upload CircuitPython UF2 file onto the bootloaded micro
       - UF2 used: **j19a_no_flash_no_crystal.uf2**
     - See here for pre-made bootloaders/UF2s
   - Test digital pin functionality by blinking a light via the CircuitPython REPL
   - Test I2C bus functionality by reading from LTR-329
   - Test SPI bus functionality by reading from BME688
9. Bringup LCD screen
   - Ensure display is set to high-speed SPI mode, attach LCD to Kairos via header link
   - Use graphics library to write sensor values to display periodically
10. Bringup SD card
    - Ensure that SD slot pins are properly soldered, with no bridges between pins and good contact between SD forks and underlying traces
    - Format SD card to FAT32, test R/W from micro to SD
    - Log sensor data to SD card

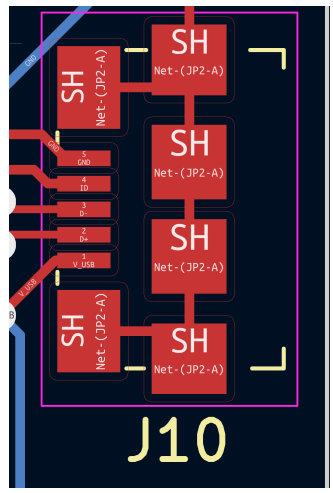
Observations

1. microUSB connection was faulty. It turns out that I used a faulty footprint which had D+ and D- flipped for the microUSB receptacle... see below:

Pin	Name	Cable color	Description
1	VCC	Red	+5 VDC
2	D-	White	Data -
3	D+	Green	Data +
4	ID		Mode Detect
5	GND	Black	Ground

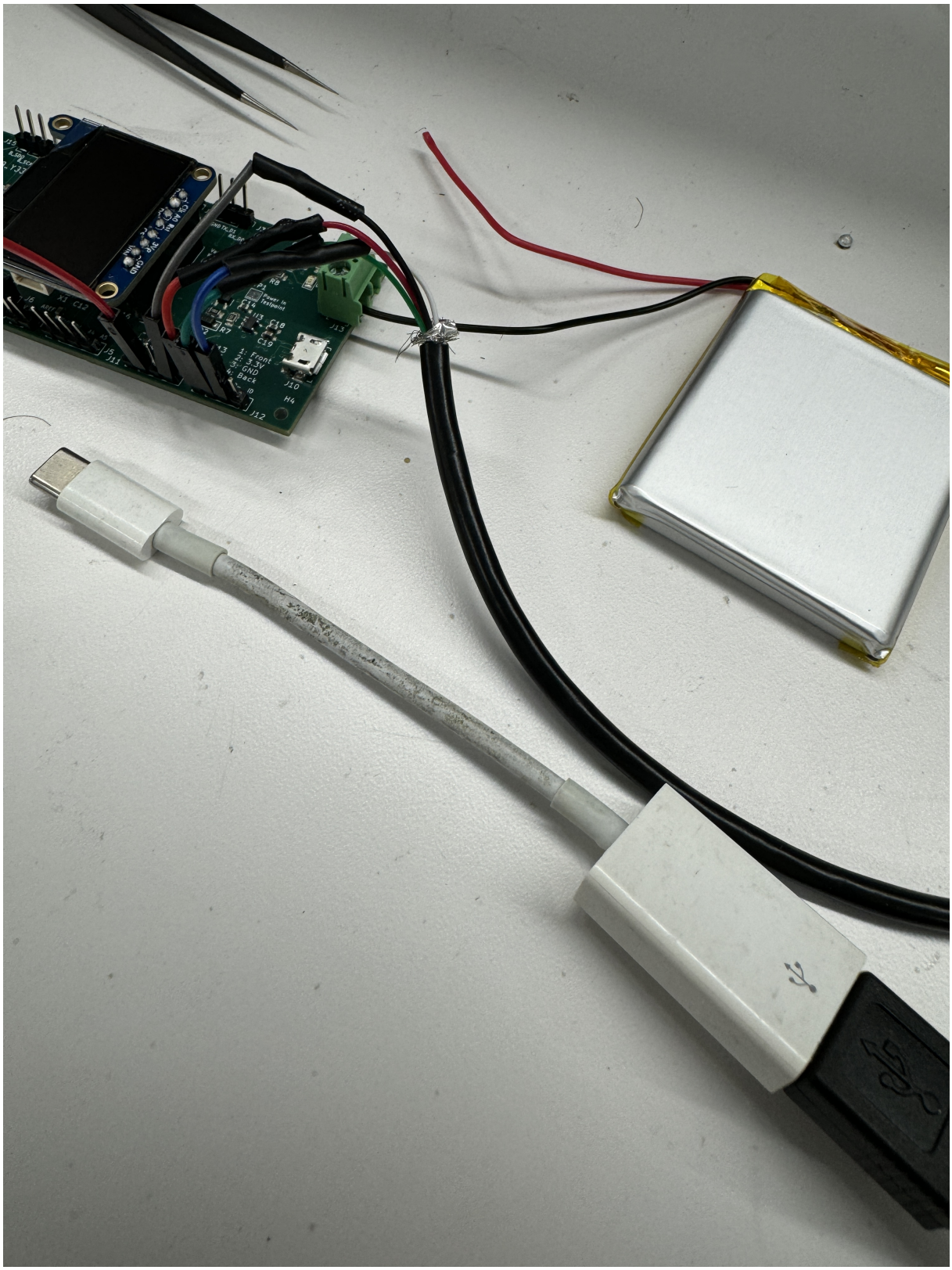
Mode Detect. May be N/C, GND or used as an attached device presence indicator (shorted to GND with resistor)

microUSB standard



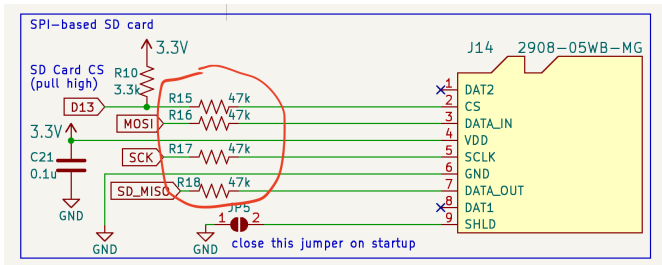
microUSB on my board

Thankfully I included headerpins with access to D+/D- elsewhere on Kairos, so I was able to use a USB to jumper cable like the one pictured below just fine.



2. SD Card was not detected given the current implementation of the circuit.

After testing a new SD card, reformatting the card to FAT32, verifying that code loaded onto the microcontroller was correct, that the SPI bus and CS line were behaving properly, and that all connections were soldered correctly I discovered that the **culprit was an issue with the the series resistors that were used**:



The circled resistors should typically be 47 Ohms instead of 47kOhms. These series resistors typically match clock speeds between the pins, but are not super critical given the similar path lengths on my board between DATA_IN, SCLK, and DATA_OUT. **Shorting R15, R16, R17, and R18 resolved the issue**

Note: FAT32 formatting was done using the following

SD Memory Card Formatter for Windows/Mac | SD Association

The SD Memory Card Formatter formats SD Memory Card, SDHC Memory Card and SDXC Memory Card (respectively SD/SDHC/SDXC Cards) complying with the SD File System Specification created by the SD Association (SDA).

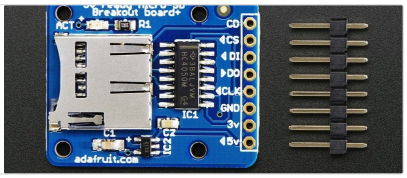
https://www.sdcard.org/downloads/formatter/

See below:

Micro SD Card Breakout Board Tutorial

How to add lots o' storage with microSD (and SD) cards

https://learn.adafruit.com/adafruit-micro-sd-breakout-board-card-tutorial/formatting-notes



3. SD Card Mounting

   Adafruit recommends something like the following for mounting an SD card using CircuitPython

```
SD_card_CS = DigitalInOut(board.PA23)
SD_card = adafruit_sdcard.SDCard(spi, SD_card_CS)
vfs = storage.VfsFat(SD_card)
storage.mount(vfs, "/sd")
```

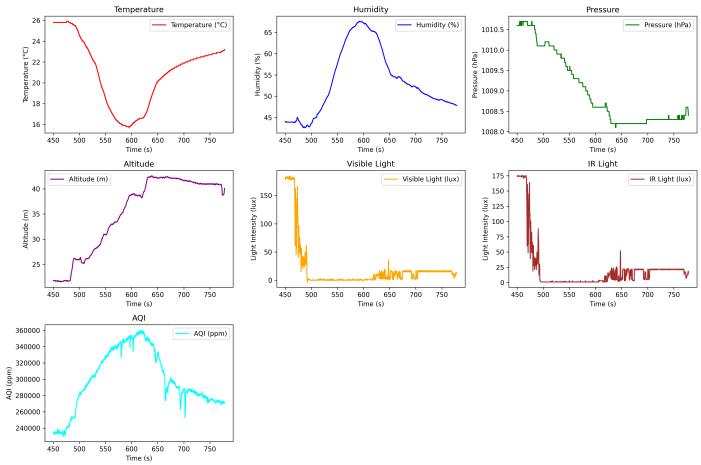
They also have this warning:

> If you are using some non-standard CircuitPython board, you may need to use specific GPIO pins for MISO and MOSI to get the SPI recognized appropriately

Initially this made caused me to manually create an "SD" folder in the SAMD's CIRCUITPYTHON directory, but that seems to create a folder conflict when mounting. **Leaving the CIRCUITPYTHON dir empty (except for the lib folder and code) does the trick.**

Data

Here's what the bike ride from the basement of Packard to my room on the second floor of West Lag looks like:



You can see that:

- It's cold outside (and humid!)
- There's some kind of positive airflow in the Packard basement labs
- There are a lot more particulates in the basement than outside. My room is almost as bad :(

Reflection

This was a great class. Even thought it took me lots of time in lab to figure out how everything came together, it ultimately did come together, and alongside it came troves of knowledge and understanding about how PCBs *actually* work. That's all to say that I feel much more comfortable designing PCBs now than before — the world is my oyster.

Bootloading microcontrollers was both the hardest and coolest part of the project. The bit of anxiety and excitement every time I tried to make my board come alive (most of which led to failures) was a thrill that kept me going.

I'm particularly proud of how portable the system became and how it all came together, especially after the complete failure of my first attempt with Stellaris. Those early mistakes taught me how to better anticipate potential issues and design with robustness in mind.

It'd be cool to keep iterating on this with some of the insights from the overview, I'm heavily considering it now that I have more PCB confidence.

Other Useful Resources/Notes

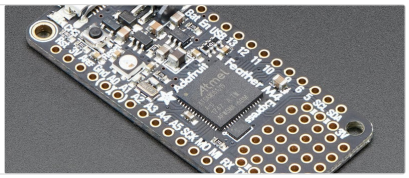
▼ Bootloader

Basics

Adafruit Feather M4 Express

This Feather is fast like a swift, smart like an owl, strong like a ox-bird

https://learn.adafruit.com/adafruit-feather-m4-express-atsamd51/uf2-bootloader-details



static.us.edusercontent.com
https://static.us.edusercontent.com/files/ycRHZwf79Parws7tc2KYLuz8

“Talking to a rock” (part II)

static.us.edusercontent.com
https://static.us.edusercontent.com/files/A2QjyuvzDjoHlHvW9BcEAqd0

How to program SAMD Bootloaders — Brent Rubell

static.us.edusercontent.com
https://static.us.edusercontent.com/files/kLr3Mi4lutUppwZj0zeLzYhb

Bootloader configuration elements that should be kept in mind:

1. Which specific version of the microcontroller you are using
2. Whether or not you have a crystal oscillator
3. Whether or not you have QSPI or SPI flash, the pins it is connected to, and whether it is currently supported by CircuitPython.
4. Which pins you would like to use. The Adafruit boards do not typically expose every pad on the microcontroller to a header pin, so in the default firmware these pins are not accessible. As suggested by Jackie, you can map all of the pins. Some of the firmware UF2s on the EE156 firmware repository have this, some do not. You must also keep in mind that certain pins are not allowed to be mapped—PA24 and PA25 on the SAME/D51, for example, are typically reserved for USB, PA00 and PA001 are reserved for the crystal, etc. If you get CircuitPython on your board, and then run into issues with accessing pins, let me know. **(Think about SERCOM assignment)!**
5. Whether or not you have a USB. If you don't have USB, you will have a hard time using CircuitPython. Although you can program your board using Arduino/C++ over JTAG, y*ou will still have more difficulty debugging without the USB*. Some students have found success cutting open a USB cable and soldering D-, D+, VUSB, and GND wires appropriately.

💡 SAMD51x UF2 readme: https://github.com/adafruit/uf2-samdx1

▼ Display

HiLetgo 0.96" SSD1306 I2C IIC SPI Serial 12864 OLED LCD Display for Adafruit Arduino Beaglebones Raspberry Pi 3.3-5V 4 Pin/7 Pin Three
Amazon.com: HiLetgo 0.96" SSD1306 I2C IIC SPI Serial 128X64 OLED LCD Display 4 Pin Font Color Blue : Electronics

https://www.amazon.com/gp/product/B01MQPQF24/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&th=1

Datasheet

cdn-shop.adafruit.com
https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf

8.1.3 MCU Serial Interface (4-wire SPI)

The 4-wire serial interface consists of serial clock: SCLK, serial data: SDIN, D/C#, CS#. In 4-wire SPI mode, D0 acts as SCLK, D1 acts as SDIN. For the unused data pins, D2 should be left open. The pins from D3 to D7, E and R/W# (WR#)# can be connected to an external ground.

Table 8-4 : Control pins of 4-wire Serial interface

Function	E(RD#)	R/W#(WR#)	CS#	D/C#	D0
Write command	Tie LOW	Tie LOW	L	L	↑
Write data	Tie LOW	Tie LOW	L	H	↑

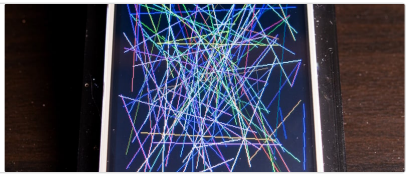
Note
(1) H stands for HIGH in signal
(2) L stands for LOW in signal

Using the SSD1306 library

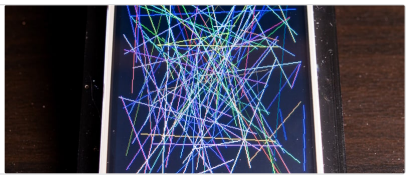
CircuitPython Hardware: SSD1306 OLED Display
How to use a SSD1306 OLED display with CircuitPython boards.
https://learn.adafruit.com/micropython-hardware-ssd1306-oled-display/circuitpython

Shapes

MicroPython Displays: Drawing Shapes
How to draw basic shapes on pixel-based displays with MicroPython!
https://learn.adafruit.com/micropython-displays-drawing-shapes/software



MicroPython Displays: Drawing Shapes
How to draw basic shapes on pixel-based displays with MicroPython!
https://learn.adafruit.com/micropython-displays-drawing-shapes/software

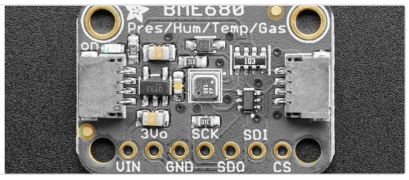


Bugs

[CircuitPython] Could not find font file font5×8.bin
https://www.reddit.com/r/raspberrypipico/comments/xvlu4c/circuitpython_could_not_find_font_file_font5×8bin/

▼ BME688

Adafruit BME680
The BME680 combines Temperature, Humidity, Pressure and Gas Sensing!
https://learn.adafruit.com/adafruit-bme680-humidity-temperature-barometic-pressure-voc-gas/



https://github.com/adafruit/Adafruit_CircuitPython_BME680

adafruit_bme680 — Adafruit BME680 Library 1.0 documentation
CircuitPython library for BME680 temperature, pressure and humidity sensor.
https://docs.circuitpython.org/projects/bme680/en/latest/api.html

▼ LTR329

Introduction — Adafruit CircuitPython LTR329 Library 1.0 documentation

Python & CircuitPython driver for LTR-329 or LTR-303 light sensors

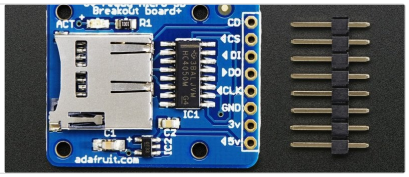
https://docs.circuitpython.org/projects/ltr329-ltr303/en/latest/

▼ SD Card

Micro SD Card Breakout Board Tutorial

How to add lots o' storage with microSD (and SD) cards

https://learn.adafruit.com/adafruit-micro-sd-breakout-board-card-tutorial/circuitpython

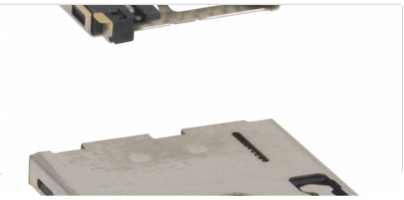


Introduction — Adafruit SD Card Library 1.0 documentation

CircuitPython driver for SD cards. This implements the basic reading and writing block functionality needed to mount an SD card using storage.VfsFat.

https://docs.circuitpython.org/projects/sd/en/latest/

2908-05WB-MG | DigiKey Electronics

Order today, ships today. 2908-05WB-MG – 8 Position Card Connector Secure Digital - microSD™ Surface Mount, Right Angle Gold from 3M. Pricing and Availability on millions of electronic components from Digi-Key Electronics.

https://www.digikey.com/en/products/detail/3m/2908-05WB-MG/1242497?s=N4IgTCBcDaIMwFkCsA2ADAdgMIBUC0AcgCIgC6AvkA