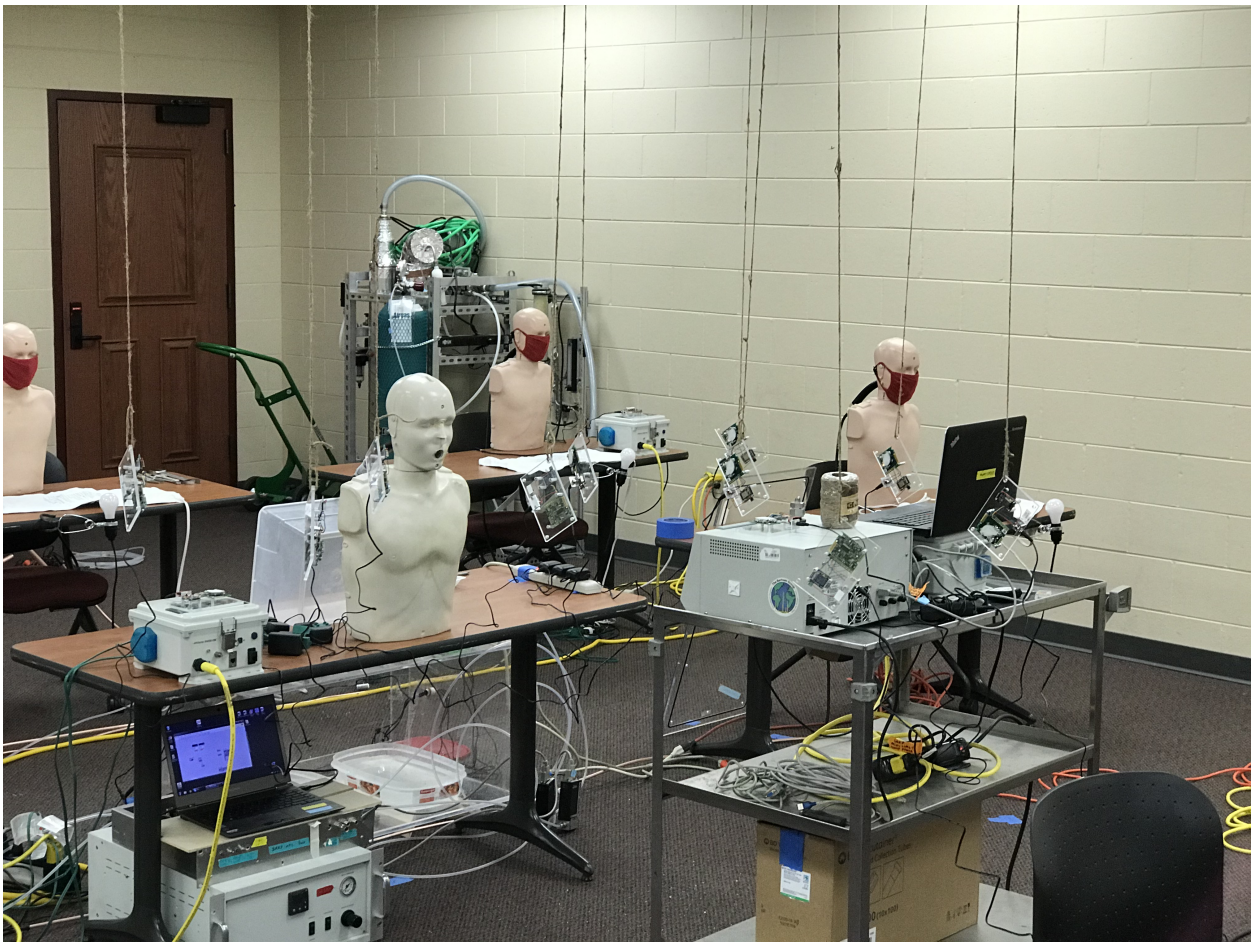
MQTT CO2 Sensor

Blaise J Thompson

December 17, 2020



1 Description

This manual documents the construction and development of the wireless, distributed CO2 sensor project. These WiFi-enabled sensors broadcast data via MQTT.

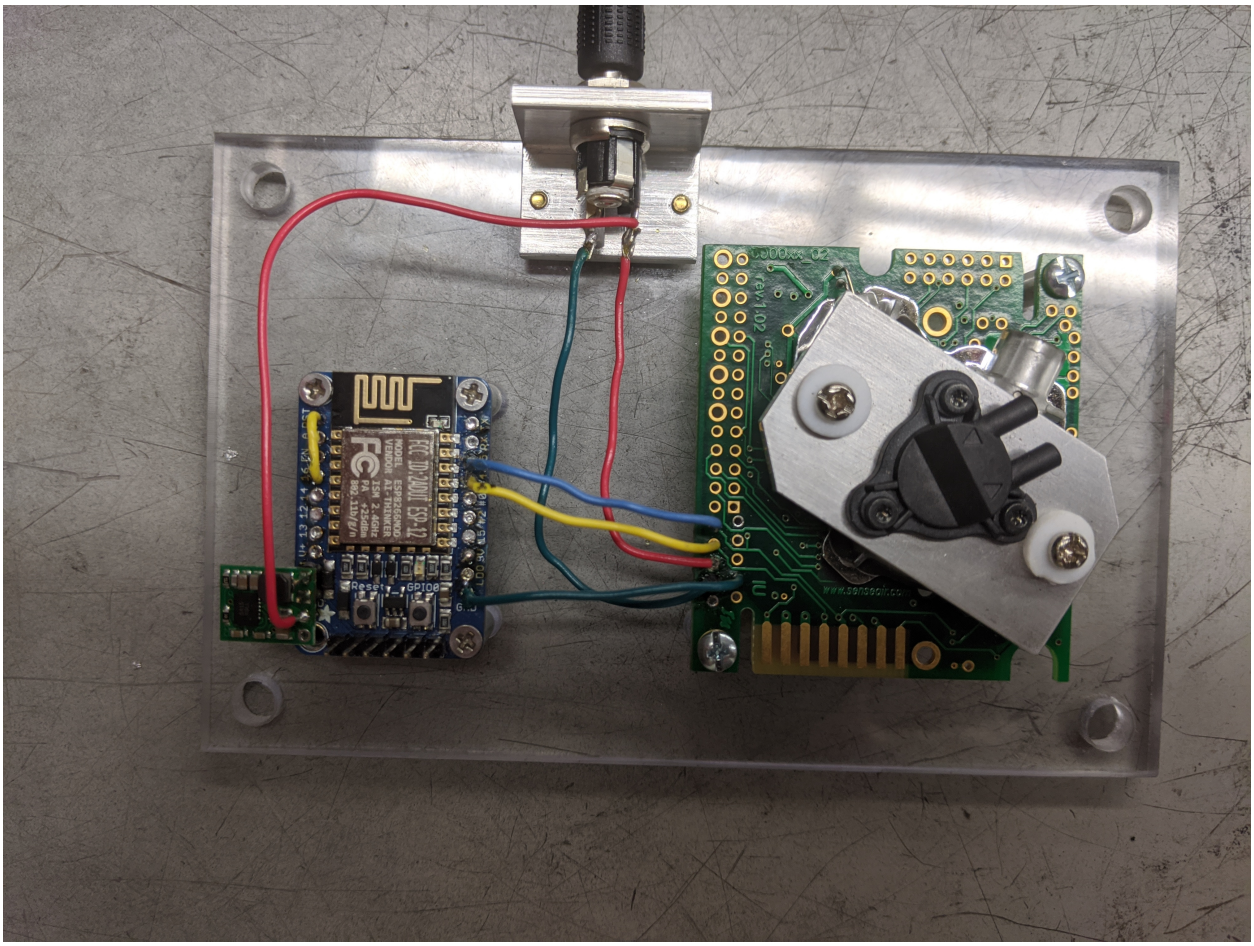
2 Wiring

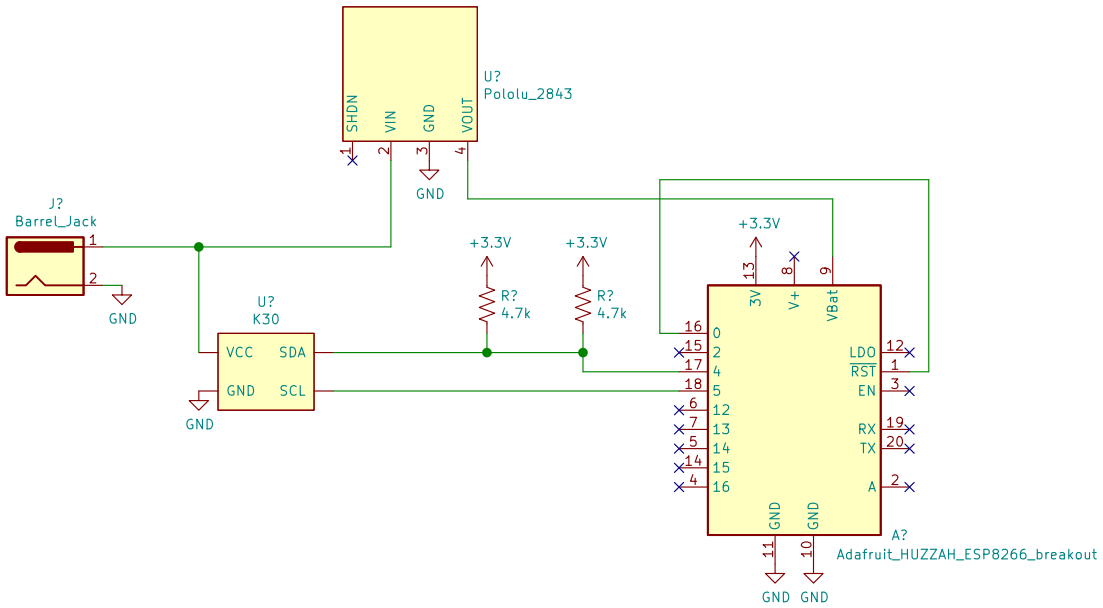
2.1 Bill of Materials

- CO2 sensor (CO2METER.COM K30 - $85)
- microcontroller (Adafruit "Huzzah" 2471 - $10)
- 5 V regulator (Pololu 2843 - $5)
- barrel jack (Switchcraft 722A - $2)
- 2x 4.7kΩ, 0.25 W resistors
- barrel jack holder, polycarbonate mounting plate (homemade)

2.2 Wiring

Wiring is pretty simple, you can see all of it here. Note that the two pull-up resistors are underneith the microcontroller. The HUZZAH break-out board has awkwardly small mounting holes, so we ended up using M2 screws.





bthompson@chem.wisc.edu
Blaise Thompson
Bertram Group
Department of Chemistry
University of Wisconsin–Madison

Sheet: /
File: circuit.sch

Title: CO2 Sensor

Size: USLetter	Date: 2020-07-28	**Rev: 1.0.0**
KiCad E.D.A. kicad 5.1.5+dfsg1-2+b1		Id: 1/1

2.3 Power Considerations

9-14 V input power (2.1 mm barrel, center positive).

32 hours battery life with 6 AA.

3 Firmware

This section describes the firmware that was flashed to the ESP8266 microcontroller. This project was built on-top of the open-source microhomie firmware, in particular microhomie-esp8266-v2.4.0-beta1.
The pre-compiled binary can be downloaded from GitHub:
https://github.com/microhomie/microhomie/releases/tag/v3.0.0-beta1.

3.1 main

main.py is automatically run by micropython when the controller initializes. In this case, we create a single HomieDevice and run it forever. Note that microhomie uses uasyncio to enable async-await syntax in micropython.

```
import settings

import sys
from machine import Pin, I2C
import network
import time
import struct

from homie.constants import FALSE, TRUE, BOOLEAN, FLOAT, STRING
from homie.device import HomieDevice
from homie.node import HomieNode
from homie.property import HomieNodeProperty
from uasyncio import get_event_loop, sleep_ms


def read_co2():
    for _ in range(10):
        try:
            i2c = I2C(scl=Pin(4), sda=Pin(5), freq=10000)
            i2c.writeto(104, "\x22\x00\x08\x2A".encode())
            time.sleep_ms(100)
            out = i2c.readfrom(104, 4)
            out = struct.unpack(">h", out[1:3])[0]
            return out
        except Exception as e:
            time.sleep_ms(250)
            print(e)


class K30(HomieNode):
    def __init__(self, name="k30", device=None):
        super().__init__(id="k30", name=name, type="sensor")
        self.device = device
        self.co2 = HomieNodeProperty(
            id="co2", name="co2", unit="ppm", settable=False, datatype=FLOAT, default=0,
        )
        self.add_property(self.co2)
        self.uptime = HomieNodeProperty(
            id="uptime", name="uptime", settable=False, datatype=STRING, default="PT0S"
        )
```

```
        self.add_property(self.uptime)
        loop = get_event_loop()
        loop.create_task(self.update_data())
        self.led = Pin(0, Pin.OUT)
        self.start = time.time()

    async def update_data(self):
        while True:
            self.led.value(0)  # on
            self.co2.data = read_co2()
            self.uptime.data = self.get_uptime()
            if self.device.mqtt.isconnected():
                self.led.value(1)  # off
            else:
                if not self.device._first_start:
                    await self.device.reset("reset")
            await sleep_ms(5000)

    def get_uptime(self):
        diff = int(time.time() - self.start)
        out = "PT"
        # hours
        if diff // 3600:
            out += str(diff // 3600) + "H"
            diff %= 3600
        # minutes
        if diff // 60:
            out += str(diff // 60) + "M"
            diff %= 60
        # seconds
        out += str(diff) + "S"
        return out


def main():
    # homie
    homie = HomieDevice(settings)
    homie.add_node(K30(device=homie))
    homie.run_forever()


if __name__ == "__main__":
    main()
```

3.2 settings

microhomie automatically reads configuration details from a file called "settings.py". Some values have been redacted as, unfortunately, login credentials for the private network cannot be shared.

```
WIFI_SSID = "chemcast"
WIFI_PASSWORD = "<REDACTED>"

MQTT_BROKER = "mqtt.chem.wisc.edu"
MQTT_USERNAME = "bertram"
MQTT_PASSWORD = "<REDACTED>"

DEVICE_ID = "bertram-co2-<ID>"
```

3.3 flash

The following bash script was used to flash the microcontroller. Importantly, the ID field must be changed for each device before flashing. This flash script uses esptool and ampy, both python utilities. Blaise also recommends rshell for debugging the microcontroller, including remote REPL and incremental file transfer.

```
# flash micropython
read -p "bring GPIO0 low, reset device, and press enter"
esptool.py --port /dev/ttyUSB0 erase_flash
read -p "bring GPIO0 low, reset device, and press enter"
esptool.py -p /dev/ttyUSB0 --baud 450800 --chip esp8266 \
           write_flash 0x00000 microhomie-esp8266-v2.4.0-beta1.bin
read -p "reset device and press enter"
# upload files
echo "sleeping"
sleep 5
echo "putting files on device"
ampy -p /dev/ttyUSB0 put main.py
ampy -p /dev/ttyUSB0 put settings.py
read -p "reset device and press enter"
echo "done!"
```

4 Server

These devices are, of course, able to publish to any MQTT broker as configured in firmware. In this case, they have been programmed to publish to https://mqtt.chem.wisc.edu. This server also runs a database application with a little bit of custom Python to link things together. All of this runs using Docker. See the git repository at https://git.chem.wisc.edu/infrastructure/mqtt for complete documentation of how this server is put together.

You may interact with the server in several different ways. These are documented on the website.