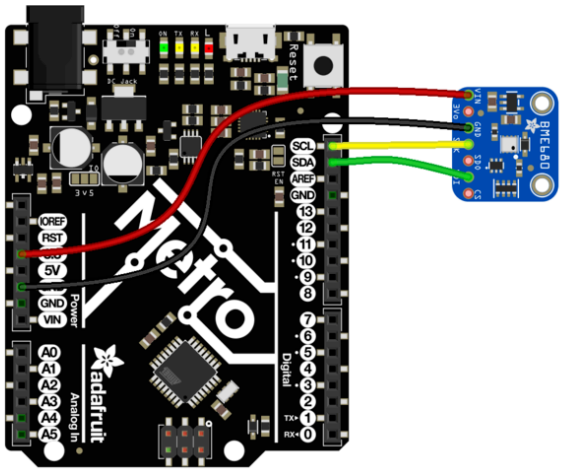
2019 GCREW CO2 Protocol (V2 Deployment)
Updated June 3, 2019 by Carmen Ritter
(Stay Tuned for 2020 updates, by Marc Rosenfield)

This protocol explains every step of the SMARTX Version 2 deployment implemented in April 2019. This version of the GCReW experiment now includes BME680 sensors as well as K30 CO2 sensors. A similar setup was used at the May deployment of the marsh organ.

The Sensors

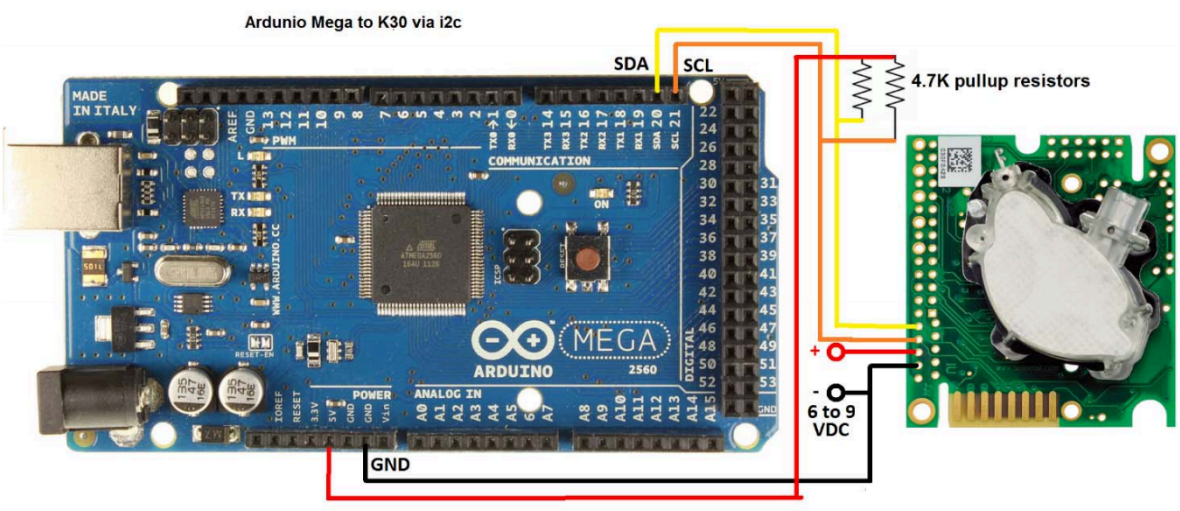
Assembling and Testing the BME680

You will need:
Arduino Mega 2560
Adafruit BME680

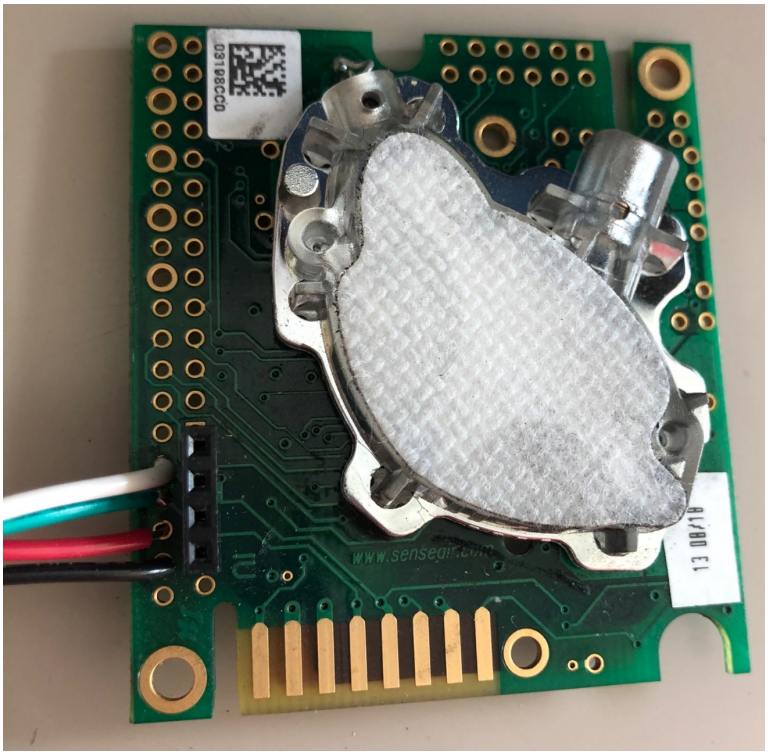


- Construct the BME680
- Cover the pins on underside and the small silver square with painters tape.
- Spray conformal coating on the BME, giving each side 2 coats with time to dry in between each.
- Connect the BME680 to the Arduino in an I2C configuration:
    - Vin to 5V
    - GND to common GND
    - SCK to SCL
    - SDI to SDA
- Download Adafruit BME680 Library
- Connect Arduino and run bmp680test in Examples to make sure sensor works
    - Also in Appendix A

Assembling and Testing the K30



You will need:
Arduino Mega 2560
K30 CO_2 Sensor
Non-solderable breadboard
1x4 stacking header
Arduino IDE
Red, white, black, green wire
NOTE: resistors shown in figure above are not necessary components



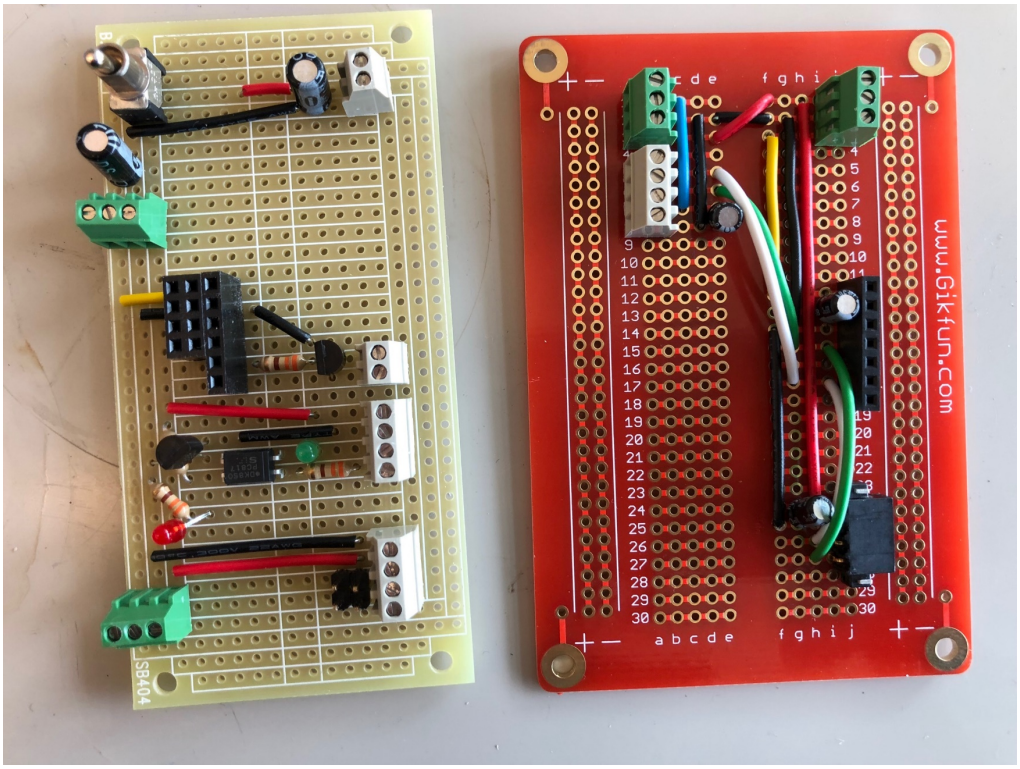
- Solder the stacking header to the bottom left inside row, leaving the bottom-most hole open.
    - This row is the UART row used for SPI connections and calibration.

- Cut a 6-inch piece of each of the four wire colors. Using wire cutters, strip 0.5cm off each end of each wire, and solder the wires to the outside row of the K30 sensor in the order shown, again excluding the bottom hole.
    - This row will be used for I2C connections.
- Cover the perimeter of the sensor (white pad) with painter's tape, then fold the tape to form a "tent" over the sensor.
    - This is so that the tape covers but does not touch the white cloth-like part of the K30.
- Conformal coat the sensor, spraying each side twice and allowing to dry in between coats.
    - Make sure the spray cannot reach the sensor's white pad.
- Calibrate the sensor. Refer to CO2 calibration protocol for instructions.

To test:

- Connect these outside wires (bottom to top):
    - GND to Arduino common GND
    - 5V to 5V
    - RXD to SCL
    - TXD to SDA
- Import and run code Arduino_to_I2C to test the K30
    - Appendix B

The Breadboards



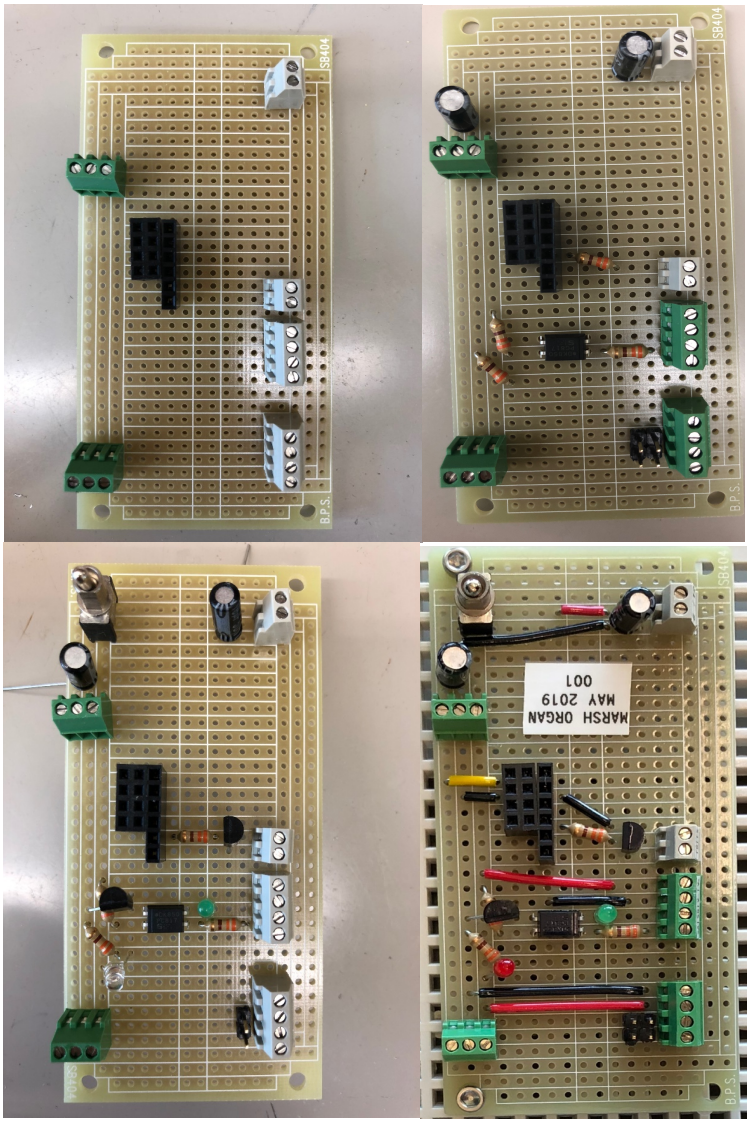
Tips:

- *Solder terminal blocks first.*
- *Solder flat wires before raised wires (shown in figure above)*
- *Clip long wires on back after each step*
- *Make sure capacitors and LED's are facing the right direction*
- *Test circuits for functionality, then conformally coat before attaching spacers, voltage regulators, or sensors*
    - *See section "Conformal Coating"*

Building the Updated Power Module

You will need:

1 SB404 breadboard
1 2x4 stacking header
1 1x6 stacking header
2 2-pin terminal blocks
2 3-pin terminal blocks
2 4-pin terminal blocks
4 330Ω resistors
2 10µF 50V capacitors
1 PC817 optocoupler
1 ALCO T11 toggle switch
1 green LED
1 red LED
2 1x2 male-to-male stacking headers
1 2222A NPN transistor

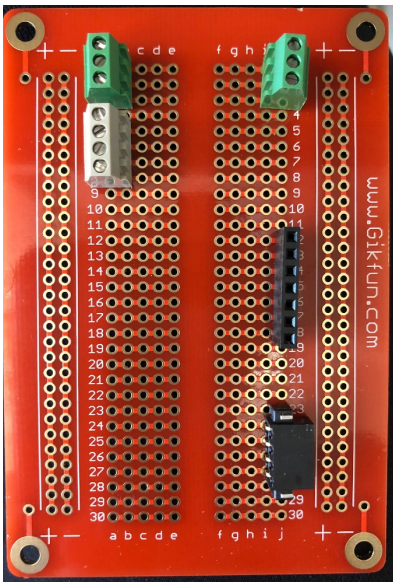
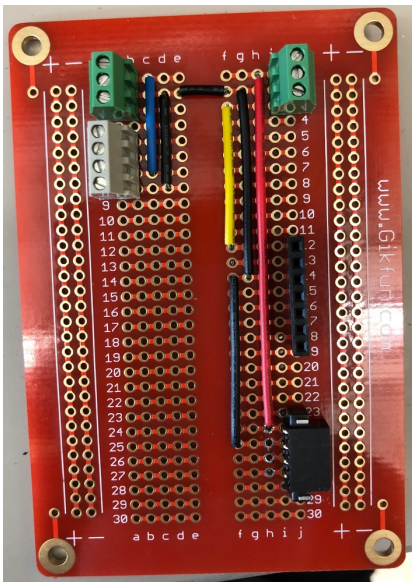
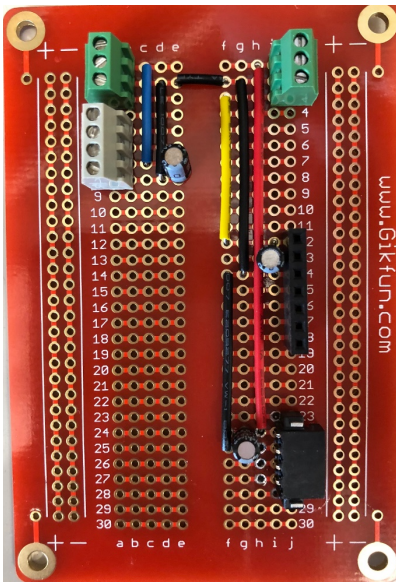
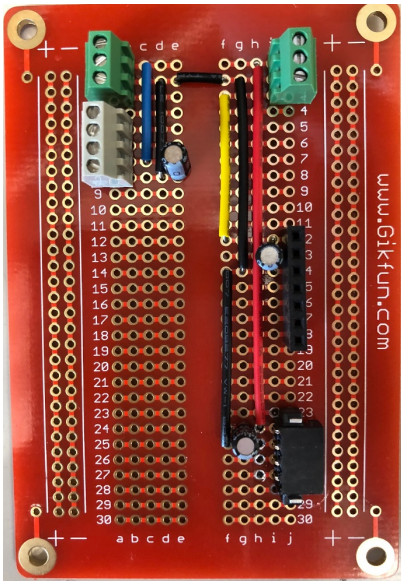
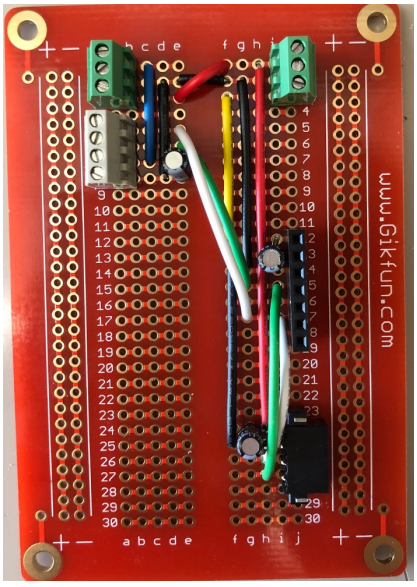
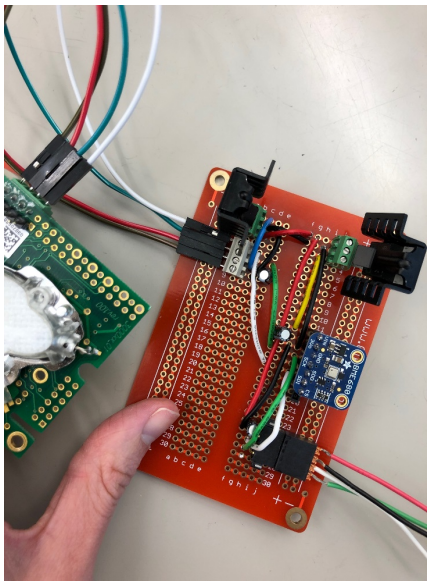


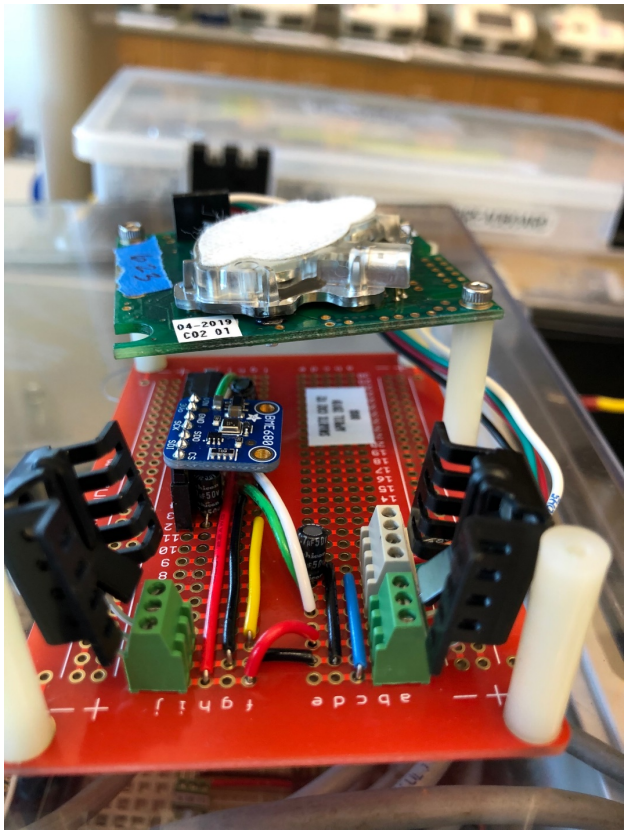
Note: top middle resistor is misplaced here. Refer to the 2nd and 4th photos for proper placement (changed in later version).

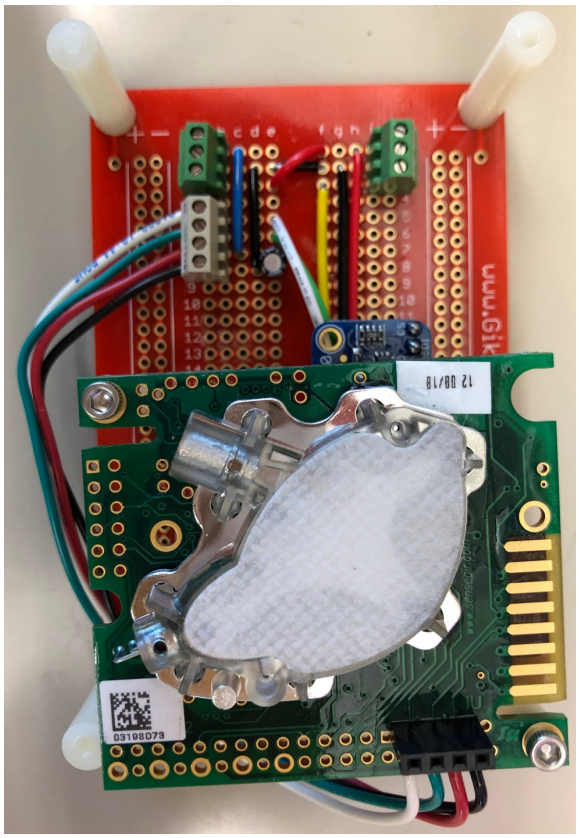
Building the Sensor Enclosure Module

You will need:

1 GikFun GK1007 breadboard
1 K30 CO2 sensor (constructed as above)
1 Adafruit BME 680 sensor (constructed as above)
1 1x7 stacking header
2 3-pin terminal blocks
1 4-pin terminal block
1 7806 ACT voltage regulator
1 7805 ACT voltage regulator
3 4.7µF 50V capacitors
2 heat sinks
1 pair 1x4 Phoenix Contact pluggable terminal blocks
5 plastic spacers
7 4/40 ¼" screws

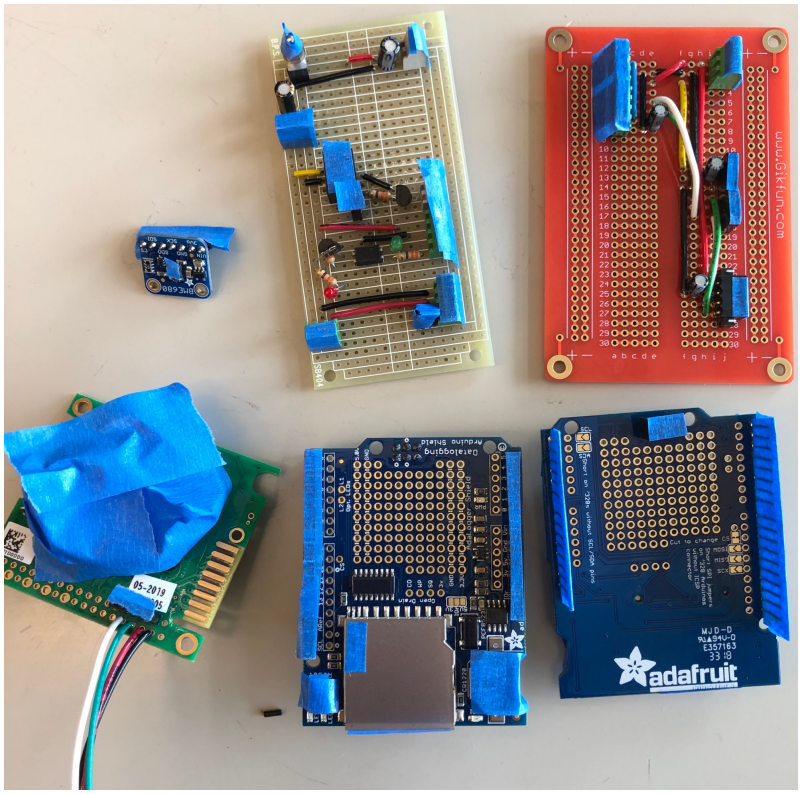








04-2019
C02 01



Conformal Coating

- Test circuits before coating them
    - If you forget, it will be very hard to move parts around
        - You will need to carefully scrape off the conformal coating with a thin blade before you'll be able to de-solder the misplaced pieces.
- Conformal coating is used to protect exposed wires and reduce the risk of water damage on circuits. It does not make a circuit entirely waterproof, but it does prevent accidental shorts and misconnections.
- Make sure all openings, moving parts, and connecting parts are covered with removable tape before beginning.
- Spray each side or surface with two coats of conformal coating
    - Hold the cannister approximately 10 inches from the surface and thinly and evenly coat it.
    - Wait until the surface is dry and non-sticky to the touch before applying a second coat
- Always make sure to spray in a well-ventilated area
    - Fume hood with cardboard beneath, or outdoors is recommended
- Conformally coat CO2 sensors before calibrating them



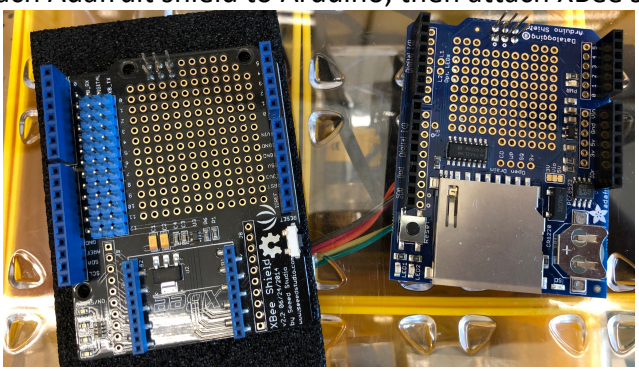
The Logger Box

You will need:
1 Bud Industries E194432 enclosure box
1 backboard to fit
1 power module
1 Adafruit Rev A shield
1 XBee v2.2 shield
1 Arduino Mega 2560
Drill and conical bit
2 cable glands
LCD screen
SPACERS
SCREWS
Micro Valve: Clippard EVR 2-12
4” zip tie

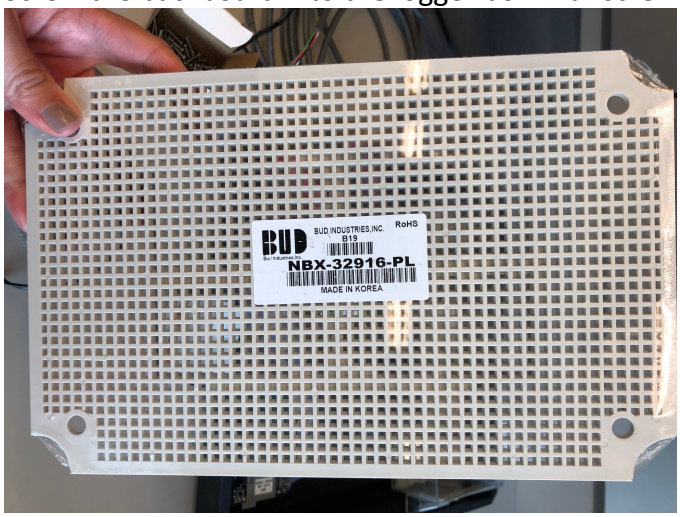
1. Drill two holes in the bottom of the enclosure box with a drill bit to match the cable gland you are using (just big enough to fit about two cables).
    a. Try to make the holes equally spaced horizontally and centered vertically.
    b. Holes should be more than an inch from the edges (horizontally).

2. Attach cable glands to box with larger piece on the outside.

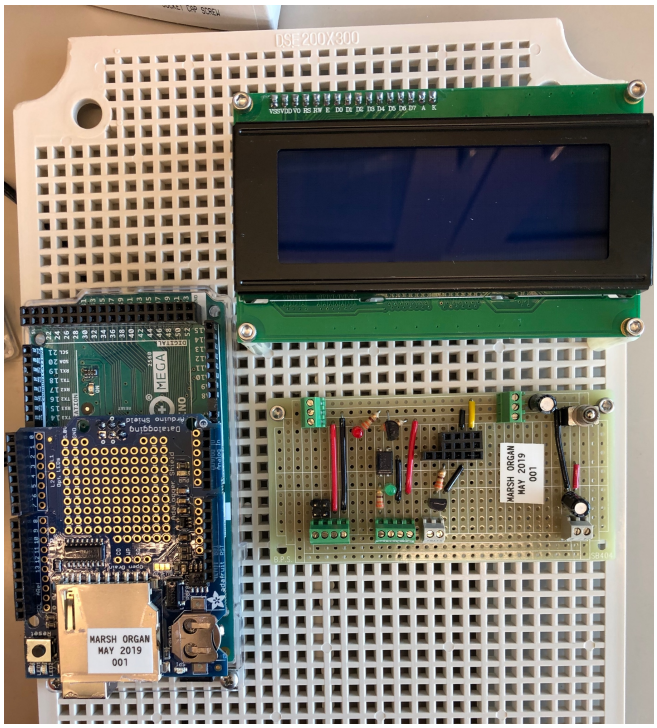
3. Attach Adafruit shield to Arduino, then attach XBee shield on top.



4. Screw the backboard into the logger box with screws provided in backboard kit.

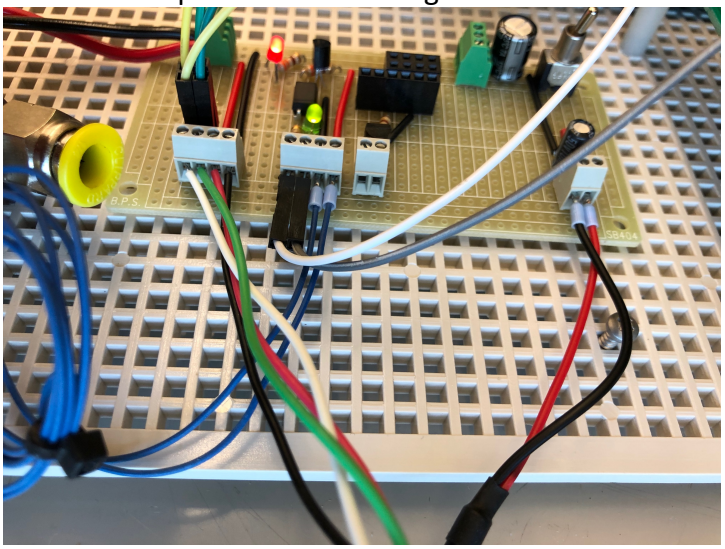


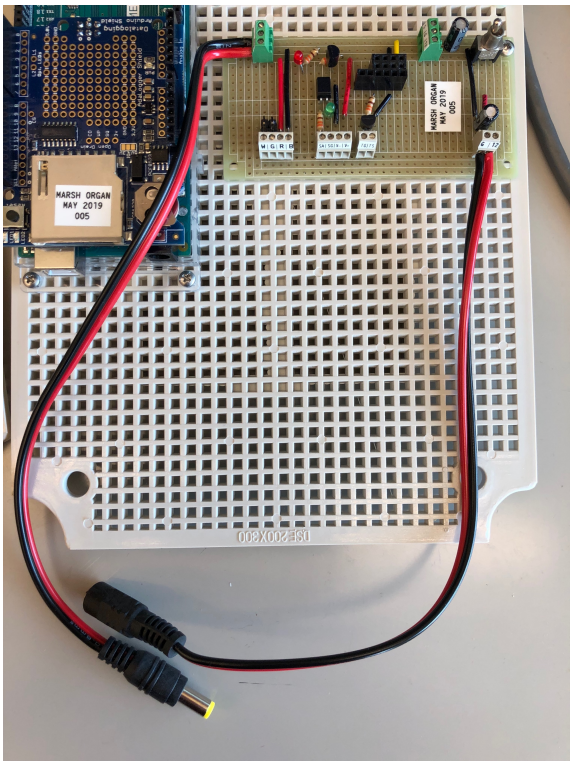
5. Attach the Arduino on its plastic back with #4 x ½ screws, the power module, and the LCD screen with spacers and screws shown in the image, into the backboard (see final picture for placement).

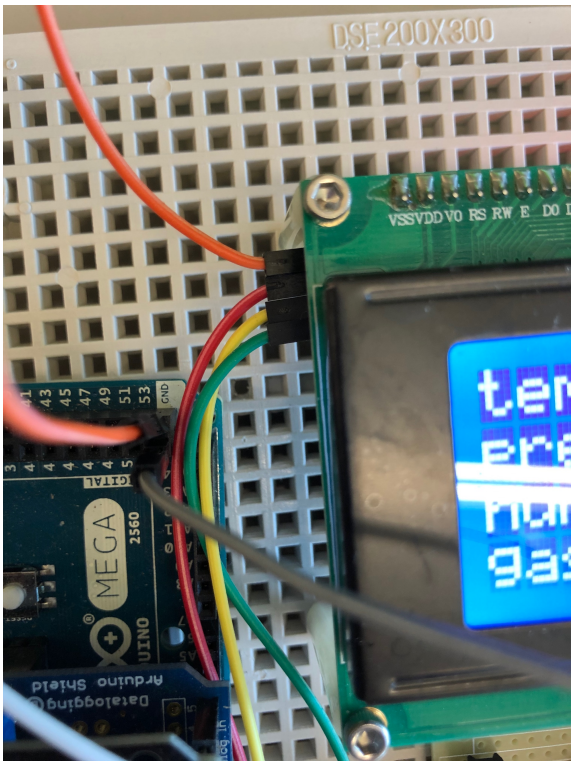


6. For each box, cut one male and one female power cord to about an 11in length. Pull apart the positive and negative ends. Strip these ends and solder them.

7. Attach wires as shown below. Zip tie loose wires together to streamline the system.

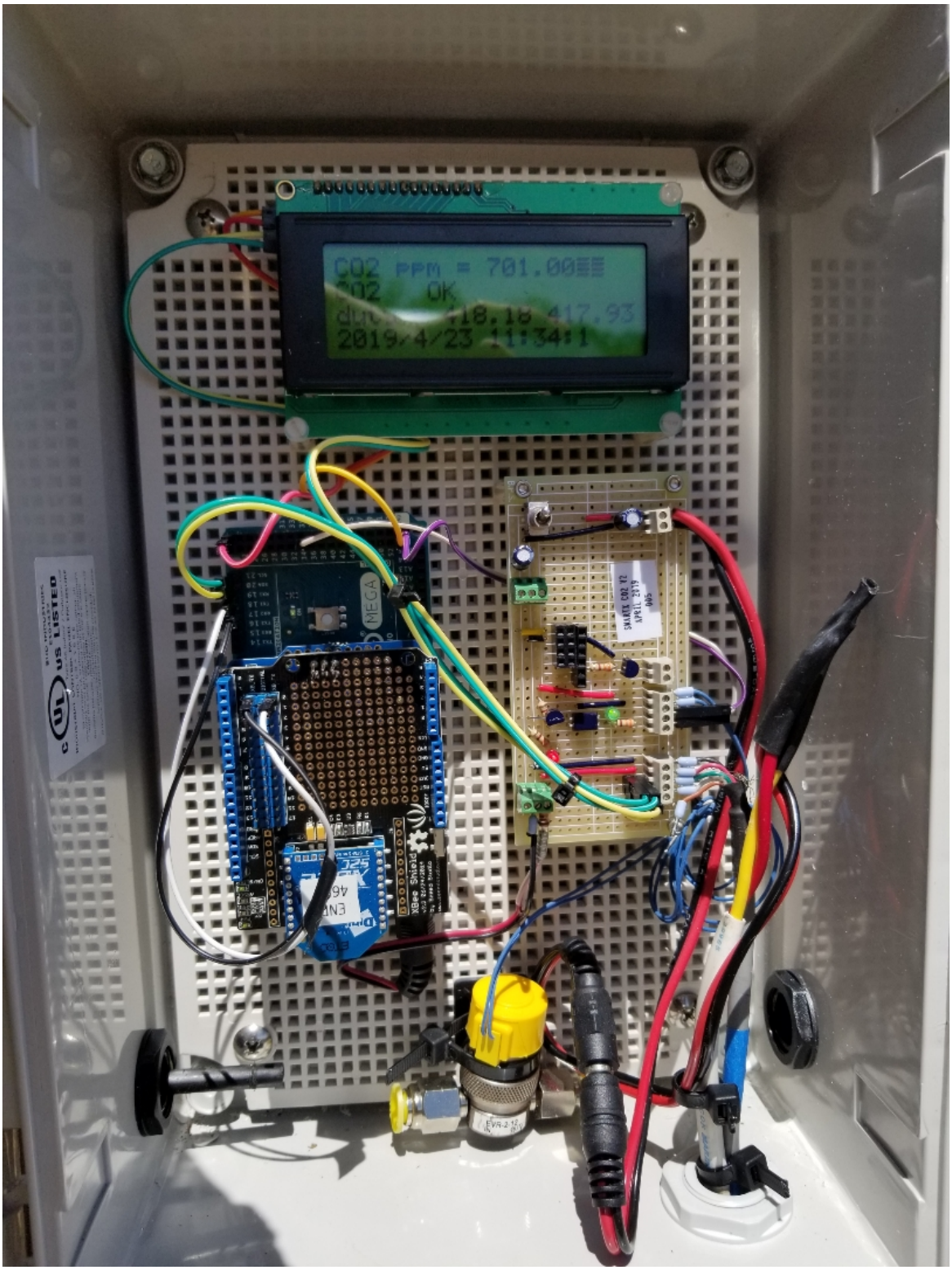






Tips

- Wear appropriate PPE when drilling box
- Don’t forget to put the SD card and battery in the shields!


CO2 PPM = 701.00
2019/4/23 11:34:1
MEGA
XBee Shield

The Hive

You will need:
1 AcuRite solar radiation shield
1 sensor enclosure module
Foot or knee high hosiery/pantyhose
2 hair ties (good quality)

1. Make a spreadsheet listing the ID number of the sensor enclosure breadboard, the BME680 sensor, the K30 sensor, as well as what plot these sensors going to. Ask a TE lab supervisor where to save the and are file. This will help keep track of what goes where.
2. Remove the bottom "layer" of the radiation shield (hive). Stretch two hair ties diagonally across the bottom layer as shown. Reattach to body of hive.
3. Carefully insert the sensor enclosure module into the pantyhose with the K30 nearest the opening.
4. Pull open the central square that the hair ties form and slip the enclosure module inside, then replace the hair ties to hold it in. This will make it easy to transport to the deployment site.

Tips

- When you pull back the rubber bands, you can usually hook them on the sides of the hive. This makes it easier for someone to do alone.

Deployment

You will need (for each plot):
AcuRite solar radiation shield
Logger box
8” solid ~14 gauge wire
1 curved PVC post (“candy cane”)
1 hive
1 small rubber band
2 4” zip ties
1 long 4-wire cable (logger box to hive in plot)

1. Wire the entire system (above).
2. Plant the PVC post inside the plot, pushed deep enough that it won’t fall over.
3. Drill the logger box into the plot’s assigned area/post.
4. Connect to power, but do not turn on.
5. Run the cable from the logger box to the plot.
    a. Be very careful not to let the exposed end of the cable touch the water.
    b. Run the cable beneath any boardwalks it may need to cross.
6. Attach the cable to the sensor enclosure module inside the hive.
7. Cover the module back up with the pantyhose, and secure the opening of the pantyhose over the cable with a rubber band.
8. Replace the enclosure module in the hive. Stretch the hair ties back over the hive entrance and zip tie the cable to the side of the hive.
    a. This should prevent the cable from being ripped out of the hive.
9. Using the thick wire, attach the hive through its handle to the PVC post.
10. Flip toggle switch in the logger box to provide power to the system. Observe the LCD screen to make sure sensors are reporting measurements.

Tips and troubleshooting

- If the screen is freezing after a period of time, try to restart the system
- If there is no connection to the sensors, make sure cable has not become unsecured.
    - Also check to see if K30 is blinking (receiving power)
- If a K30 sensor is measuring unnaturally high CO2 levels, it’s likely the sensor was conformally coated after being calibrated. It needs recalibration.

ADD CODE APPENDICES

Appendix A

```
/**************************************************************************
  This is a library for the BME680 gas, humidity, temperature & pressure sensor

  Designed specifically to work with the Adafruit BME680 Breakout
  ----> http://www.adafruit.com/products/3660

  These sensors use I2C or SPI to communicate, 2 or 4 pins are required
  to interface.

  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing products
  from Adafruit!

  Written by Limor Fried & Kevin Townsend for Adafruit Industries.
  BSD license, all text above must be included in any redistribution
 **************************************************************************/

#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include "Adafruit_BME680.h"

#define BME_SCK 13
#define BME_MISO 12
#define BME_MOSI 11
#define BME_CS 10

#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BME680 bme; // I2C
//Adafruit_BME680 bme(BME_CS); // hardware SPI
//Adafruit_BME680 bme(BME_CS, BME_MOSI, BME_MISO,  BME_SCK);

void setup() {
  Serial.begin(9600);
  while (!Serial);
  Serial.println(F("BME680 test"));

  if (!bme.begin()) {
    Serial.println("Could not find a valid BME680 sensor, check wiring!");
    while (1);
  }
```

```
  // Set up oversampling and filter initialization
  bme.setTemperatureOversampling(BME680_OS_8X);
  bme.setHumidityOversampling(BME680_OS_2X);
  bme.setPressureOversampling(BME680_OS_4X);
  bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
  bme.setGasHeater(320, 150); // 320*C for 150 ms
}

void loop() {
  if (! bme.performReading()) {
    Serial.println("Failed to perform reading :(");
    return;
  }
  Serial.print("Temperature = ");
  Serial.print(bme.temperature);
  Serial.println(" *C");

  Serial.print("Pressure = ");
  Serial.print(bme.pressure / 100.0);
  Serial.println(" hPa");

  Serial.print("Humidity = ");
  Serial.print(bme.humidity);
  Serial.println(" %");

  Serial.print("Gas = ");
  Serial.print(bme.gas_resistance / 1000.0);
  Serial.println(" KOhms");

  Serial.print("Approx. Altitude = ");
  Serial.print(bme.readAltitude(SEALEVELPRESSURE_HPA));
  Serial.println(" m");

  Serial.println();
  delay(2000);
}
```

Appendix B

```
// CO2 Meter K-series Example Interface
// Revised by Marv Kausch, 7/2016 at CO2 Meter <co2meter.com>
// Talks via I2C to K30/K22/K33/Logger sensors and displays CO2 values
// 12/31/09
#include <Wire.h>
// We will be using the I2C hardware interface on the Arduino in
// combination with the built-in Wire library to interface.
// Arduino analog input 5 - I2C SCL
// Arduino analog input 4 - I2C SDA
/*
  In this example we will do a basic read of the CO2 value and checksum verification.
  For more advanced applications please see the I2C Comm guide.
*/
int co2Addr = 0x68;
// This is the default address of the CO2 sensor, 7bits shifted left.
void setup() {
  Serial.begin(9600);
  Wire.begin ();
  pinMode(13, OUTPUT); // address of the Arduino LED indicator
  Serial.println("Application Note AN-102: Interface Arduino to K-30");
}
////////////////////////////////////////////////////////////////
// Function : int readCO2()
// Returns : CO2 Value upon success, 0 upon checksum failure
// Assumes : - Wire library has been imported successfully.
// - LED is connected to IO pin 13
// - CO2 sensor address is defined in co2_addr
////////////////////////////////////////////////////////////////
int readCO2()
{
  int co2_value = 0;  // We will store the CO2 value inside this variable.

  digitalWrite(13, HIGH);  // turn on LED
  // On most Arduino platforms this pin is used as an indicator light.

  //////////////////////////
  /* Begin Write Sequence */
  //////////////////////////

  Wire.beginTransmission(co2Addr);
  Wire.write(0x22);
  Wire.write(0x00);
  Wire.write(0x08);
```

```
Wire.write(0x2A);

Wire.endTransmission();

//////////////////////
/* End Write Sequence. */
//////////////////////

/*
  We wait 10ms for the sensor to process our command.
  The sensors's primary duties are to accurately
  measure CO2 values. Waiting 10ms will ensure the
  data is properly written to RAM

*/

delay(10);

//////////////////////
/* Begin Read Sequence */
//////////////////////

/*
  Since we requested 2 bytes from the sensor we must
  read in 4 bytes. This includes the payload, checksum,
  and command status byte.

*/

Wire.requestFrom(co2Addr, 4);

byte i = 0;
byte buffer[4] = {0, 0, 0, 0};

/*
  Wire.available() is not nessessary. Implementation is obscure but we leave
  it in here for portability and to future proof our code
*/
while (Wire.available())
{
  buffer[i] = Wire.read();
  i++;
}
```

```
  //////////////////////
  /* End Read Sequence */
  //////////////////////

  /*
    Using some bitwise manipulation we will shift our buffer
    into an integer for general consumption
  */

  co2_value = 0;
  co2_value |= buffer[1] & 0xFF;
  co2_value = co2_value << 8;
  co2_value |= buffer[2] & 0xFF;


  byte sum = 0; //Checksum Byte
  sum = buffer[0] + buffer[1] + buffer[2]; //Byte addition utilizes overflow

  if (sum == buffer[3])
  {
    // Success!
    digitalWrite(13, LOW);
    return co2_value;
  }
  else
  {
    // Failure!
    /*
      Checksum failure can be due to a number of factors,
      fuzzy electrons, sensor busy, etc.
    */

    digitalWrite(13, LOW);
    return 0;
  }
}
void loop() {

  int co2Value = readCO2();
  if (co2Value > 0)
  {
    Serial.print("CO2 Value: ");
    Serial.println(co2Value);
  }
```

```
  else
  {
    Serial.println("Checksum failed / Communication failure");
  }
  delay(2000);
}
```

Appendix F

GCREWco2controlv2019v_I2C_v4

```
// Code for C02 logger test modules
// Roy Rich (based on sample code from several online sources in Arduino packages used here. Code does 1) measuress Co2, calculates duty cycle and opens valve, stores data in SD card, enables modbus communication
// Uses a variety of open source code from Arduino
// Reports values from a K-series sensor back to the computer
// written by Jason Berger
// Co2Meter.com
//
// 5-25-2018
// V.1.5 Adjustments
// faster over co2 reduction
// redo nightime co2 stoppage
// 4-15-2019
// I2C_V0 Carmen Ritter and Roy Rich
// Code integrates I2C control of K30 unit
// 4-16-2019
// I2C_V1 Carmen Ritter and Roy Rich
// I2C control of BME680 sensor: T(*C), Pressure(mbar), Humidity(%), Gas(KOhms), Approx. Alt(m) (based on defined SEALEVEL

/*
  SimpleModbusSlaveV10 supports function 3, 6 & 16.

  This example code will receive the adc ch0 value from the arduino master.
  It will then use this value to adjust the brightness of the led on pin 9.
  The value received from the master will be stored in address 1 in its own
  address space namely holdingRegs[].

  In addition to this the slaves own adc ch0 value will be stored in
  address 0 in its own address space holdingRegs[] for the master to
  be read. The master will use this value to alter the brightness of its
  own led connected to pin 9.

  The modbus_update() method updates the holdingRegs register array and checks
  communication.

  Note:
  The Arduino serial ring buffer is 64 bytes or 32 registers.
  Most of the time you will connect the arduino to a master via serial
  using a MAX485 or similar.
```

```
  In a function 3 request the master will attempt to read from your
  slave and since 5 bytes is already used for ID, FUNCTION, NO OF BYTES
  and two BYTES CRC the master can only request 58 bytes or 29 registers.

  In a function 16 request the master will attempt to write to your
  slave and since a 9 bytes is already used for ID, FUNCTION, ADDRESS,
  NO OF REGISTERS, NO OF BYTES and two BYTES CRC the master can only write
  54 bytes or 27 registers.

  Using a USB to Serial converter the maximum bytes you can send is
  limited to its internal buffer which differs between manufactures.
*/
////////////////////////////////////////////////////////////////////////////////////
// CODE BEGINS HERE //
////////////////////////////////////////////////////////////////////////////////////

//I2C comment
//#include <kSeries.h> // include kSeries Library
#include <Wire.h>
#include <LiquidCrystal_I2C.h> // LCD display
#include <SPI.h>
#include <SD.h> // SD card
#include <Average.h> // Data averages
#include <RTClib.h> // Real Time Clock
#include <SimpleModbusSlave.h> // Modbus
#include <EEPROM.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME680.h>

#define SEALEVELPRESSURE_HPA (1013.25) // to calibrate BME680 altitude
#define BACKLIGHT_PIN (3) // keep (not sure for reason)
#define LED_ADDR (0x27) // might need to be 0x3F, if 0x27 doesn't work
#define LED 41 // for modbus testing

//declare lcd, rtc, k30 address, sd chipselect, bme680
LiquidCrystal_I2C lcd(LED_ADDR, 2, 1, 0, 4, 5, 6, 7, BACKLIGHT_PIN, POSITIVE) ;
char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday"};
RTC_PCF8523 rtc;
//I2C comment
//kSeries K_30(50,51); //Initialize a kSeries Sensor with pin 50 as Rx and 51 as Tx
int co2Addr = 0x34; // new address of CO2 sensor, 7 bits shifted left
const int chipSelect=10;
```

```
Adafruit_BME680 bme; //BME 680 sensor

int lcd_state;
int lcd_state2;

///SET THESE FOR USER
unsigned long intervalco2= 5000; // set for measurement interval for k30 sensor in milliseconds
File dataFile;//datafile name for co2 logger sd chip
int sl_id = EEPROM.read(0);
const double treatment = 750; // set for co2 target
double duty0 = 300; // set for intial duty cycle starting point
const double co2slope = 4; // set for intial duty cycle starting point; controls rate of change,
smaller = slower
const int Pin34 = 34;


//const int Pin52 = 52;// ??Commented out for 2019
const int Pin53 = 53;// ??

int flag;
int State1 = LOW; //valve state closed
unsigned long previousMillis1 = 0;
unsigned long previousMillis2 = 0;
unsigned long previousMillis3 = 0;
unsigned long previousMillis4 = 0;
unsigned long previousMillis5 = 0;
unsigned long previousMillis6 = 0;
unsigned long previousMillis7 = 0;
unsigned long previousMillis8 = 0;
unsigned long previousMillis9 = 0;
unsigned long previousMillis10 = 0;
unsigned long previousMillis11 = 0;
unsigned long previousMillis12 = 0;

double interval1 = 500;// period length for valve
double duty1; // interval open for valve
double dutyp; // previous inteval for valve
double co2; // k30reported c02
double pco2;// previous co2 (tminus 3 secs)
int hr; // hours
float co2ave1; //= co2ave.push(co2);
float co2ave5; //= co2ave.push(co2);
float dutyave5; // = dutyave.push(duty1);
//float co2sd5; //= co2sd.push(co2);
```

```
// from CSlogger
float time_cs;
float dutycs;
float CS_Code= 0;

//BME680variable
double temp; //*C
double pressure; //mbar
double humidity; //%
double gas; //KOhms
double approx_alt; //m


Average <float> co2_5(60); // 5 minute average
Average <float> co2_1(12); // 1 minute average
Average <float> duty_5(60);// 5 minute average

// for modbus
enum
{
  aa,// co2 value (TO CS)
  ab,
  ba,// duty1 value (TO CS)
  bb,
  ca,// co2ave2 (TO CS) up to 2018, 2019 = gas
  cb,
  da,// co2ave5(TO CS)
  db,
  ea,// dutyave5(TO CS)
  eb,
  fa,// co2sd5(TO CS)
  fb,
  ga,// Open(TO CS)2018; pressure 2019
  gb,
  ha,// time_CS (from CS)
  hb,
  ia,// dutyconstant(from CS)
  ib,
  ja,// open(from CS)
  jb,
  ka,// CS_code(from CS)
  kb,
  la,// open(from CS)
```

```
  lb,
  ma,// open(from CS)
  mb,
  HOLDING_REGS_SIZE// leave this one
  // total number of registers for function 3 and 16 share the same register array
  // i.e. the same address space
};

// variable encoding for modbus
uint16_t holdingRegs[HOLDING_REGS_SIZE]; // function 3 and 16 register array
union Pun {float f; uint32_t u;};

// for variables to logger
void encodeFloat(uint16_t *holdingRegs, float x)
{
  union Pun pun;
  pun.f = x;
  holdingRegs[0] = (pun.u >> 16) & 0xFFFFU;
  holdingRegs[1] = pun.u & 0xFFFFU;
}

// for varaibles from logger
float decodeFloat(const uint16_t *regs)
{
  union Pun pun;
  pun.u = ((uint32_t)regs[0] << 16) | regs[1];
  return pun.f;
}

void software_Reset() // Restarts program from beginning but does not reset the peripherals
and registers
{
  asm volatile ("  jmp 0");
}

//added 5-2019 alternative resets
//void softwareReset( uint8_t prescaller) {
//  // start watchdog with the provided prescaller
//  wdt_enable( prescaller);
//  // wait for the prescaller time to expire
//  // without sending the reset signal by using
//  // the wdt_reset() method
//  while(1) {}
//}
```

```
//void softwareReset2 (unsigned long delayMillis)
//{
//  uint32_t resetTime = millis() + delayMillis;
//  while ( resetTime > millis()) { /* wait and do nothing... */}
//  // set digital pin 7 to LOW - reset the MCU
//  digitalWrite(31, LOW);
//}

// Function: void setup()
// Returns: void
// Assumes: libraries are included, sensors connected via I2C
// Initializes dataFile on SD, LCD, BME680, k30
// Configures modbus communications

void setup()
{
  pinMode(34, OUTPUT);
  digitalWrite(34, LOW);

  //modbus
  /* parameters(HardwareSerial* SerialPort,
            long baudrate,
    unsigned char byteFormat,
            unsigned char ID,
            unsigned char transmit enable pin,
            unsigned int holding registers size,
            unsigned int* holding register array)
    Valid modbus byte formats are:
    SERIAL_8N2: 1 start bit, 8 data bits, 2 stop bits
    SERIAL_8E1: 1 start bit, 8 data bits, 1 Even parity bit, 1 stop bit
    SERIAL_8O1: 1 start bit, 8 data bits, 1 Odd parity bit, 1 stop bit

    You can obviously use SERIAL_8N1 but this does not adhere to the
    Modbus specifications. That said, I have tested the SERIAL_8N1 option
    on various commercial masters and slaves that were suppose to adhere
    to this specification and was always able to communicate... Go figure.
    These byte formats are already defined in the Arduino global name space.
  */
  // void modbus_configure(long baud, unsigned char _slaveID, unsigned char _TxEnablePin,
unsigned int _holdingRegsSize, unsigned char _lowLatency)
  //Serial1.begin(9600)/ ports 18, 19 on arduino; for loggernet communication

  modbus_configure(&Serial1, 9600, SERIAL_8N1,sl_id, 41, HOLDING_REGS_SIZE, holdingRegs);
```

```
  // modbus_update_comms(baud, byteFormat, id) is not needed but allows for easy update of
the
  // port variables and slave id dynamically in any function.
  modbus_update_comms(9600, SERIAL_8N1, sl_id);

  //Serial.begin(9600); // this is for usb com port
  Wire.begin(); // for sensors
  lcd.begin(20,4);

 //While (!Serial); //do nothing (implemented with I2C sensors) // not need for arduino mega

//  if (! rtc.begin())
//  {
//    Serial.println("Couldn't find RTC"); /// real time clock
//    while (1);
//  }
//  if (! rtc.isrunning())
//  {
//    Serial.println("RTC is NOT running!");
//    // following line sets the RTC to the date & time this sketch was compiled
// rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
//    // This line sets the RTC with an explicit date & time, for example to set
//    // January 21, 2014 at 3am you would call:
//    // rtc.adjust(DateTime(2014, 1, 21, 3, 0, 0));
//  }

//! rtc.initialized()


  //sd chip  code
//  Serial.print("Initializing SD card...");
  // make sure that the default chip select pin is set to
  // output, even if you don't use it:

pinMode(SS, OUTPUT);
  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect)) {
    lcd.clear();
    lcd.setCursor (0, 1 );       // go to the next line
    lcd.print("Card Fail");
    // don't do anything more:
    while (1) ;
  }
  if (SD.begin(chipSelect)) {
```

```
  lcd.clear();
  lcd.setCursor (0, 1);     // go to the next line
  lcd.print("CARD PASS");
 }

 // Open up the file we're going to log to
 dataFile = SD.open("datalog.txt", FILE_WRITE);
 if (! dataFile) {
  lcd.setCursor(0, 2);
  lcd.println("error datalog");
  // Wait forever since we cant write data
  while (1);
 }
  if (dataFile) {
  lcd.setCursor(0, 2);
  lcd.print("datalog cool");
 }

 //BME 680 setup
 if (!bme.begin())
 {
  lcd.setCursor(0, 3);
  lcd.print("BME680 fail");
  //while (1); // Do nothing //TODO do we want the program to continue if one sensor fails like
this?
 }
 if (bme.begin())
 {
  lcd.setCursor(0, 3);
  lcd.print("BME680 pass");
  //while (1); // Do nothing //TODO do we want the program to continue if one sensor fails like
this?
 }
 delay(1000);

 //write header to dataFile on SD removed for repeated groups
 //dataFile.println("unixtime, timestamp, slave_ID, co2, duty1, co2ave1, co2ave5, dutyave5,
co2sd5, temp, pressure, humidity, gas, approx_altitude");

 // sd code

 //Serial.println(" AN-216 This uses the kSeries.h library");

 // Switch on the lcd and lcd backlight
```

```
  pinMode (BACKLIGHT_PIN, OUTPUT );
  digitalWrite (BACKLIGHT_PIN, HIGH );
  //lcd.begin(20,4); // initialize the lcd
  lcd.home (); // go home
  lcd.print("CO2 Start Up");
  lcd.setCursor ( 0, 1 ); // go to the next line
  lcd.print ("RRich v1.8");
  lcd.setCursor ( 0, 2 ); // go to the next line
  lcd.print ("Gary is my Hero!");
  delay(1000);

  pinMode(34, OUTPUT); // channel for clippard unit
  //pinMode(52, OUTPUT); // indicator light for co2 sample
  pinMode(53, OUTPUT); // indicator light for sd
  //pinMode(13, OUTPUT); //indicator for k30

  //BME 680 setup
//if (!bme.begin()) {
   //Serial.println("Could not find a valid BME680 sensor, check wiring!");
//   //while (1); // Do nothing //TODO do we want the program to continue if one sensor fails
like this?
//}
//
  // Set up oversampling and filter initialization

  bme.setTemperatureOversampling(BME680_OS_8X);
  bme.setHumidityOversampling(BME680_OS_2X);
  bme.setPressureOversampling(BME680_OS_4X);
  bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
  bme.setGasHeater(320, 150); // 320*C for 150 ms
  }

// Function: int readCO2()
// Returns: CO2 Value upon success, 0 upon checksum failure
// Assumes: Wire library has been imported correctly
// LED connected to IO pin 13
// CO2 sensor address defined in co2_addr

int readCO2()
{
  int co2_value = 0; // co2 value stored here
  //digitalWrite(13, HIGH); // turn on LED removed v2

  // Write sequence
```

```
Wire.beginTransmission(co2Addr);
Wire.write(0x22);
Wire.write(0x00);
Wire.write(0x08);
Wire.write(0x2A);
Wire.endTransmission();

// waiting 10ms is necessary for accurate data - K30 has time to process
// TODO: MAKE TIMING LOOP if this messes up rest of code
delay(10);

// Read sequence
// Requested 2 bytes from sensor, so read in 4 bytes
Wire.requestFrom(co2Addr, 4);

byte i = 0;
byte buffer[4] = {0, 0, 0, 0};

// Wire.available() not necessary, but makes code safer
while (Wire.available())
{
  buffer[i] = Wire.read();
  i++;
}

// bitwise manipulation to make co2 value readable
// |= is compound bitwise "OR" function
co2_value = 0;
co2_value |= buffer[1] & 0xFF;
co2_value = co2_value << 8;
co2_value |= buffer[2] & 0xFF;

byte sum = 0; // checksum byte
sum = buffer[0] + buffer[1] + buffer[2];

if (sum == buffer[3])
{
  // SUCCESS
  //digitalWrite(13, LOW); // changes in this version 3
  return co2_value;
}

else
{
```

```
   // FAILURE : checksum failure possibly due to static, sensor busy, etc.
   //digitalWrite(13, LOW); removed v2
   return 0;
  }
}

// Function: void loop()
// Returns: void
// Assumes: above code works
// collects data and prints to file, LCD, and serial
// averages values collected over certain times
// TODO: do we want to have everything collecting at the same time? if so, should we have
// avg and stdev code set up for each of the five BME values gathered?

void loop()
{
  unsigned long currentMillis10 = millis(); // collection every 3 seconds
  if (currentMillis10 - previousMillis10 >= 7200000)
  {
    // resets every 2 hours
  software_Reset();
  previousMillis10 = currentMillis10;
  }

  unsigned long currentMillis8 = millis(); // collection every 3 seconds

  if (currentMillis8 - previousMillis8 >= 60000)
  {
    // resets every 2 hours
    //function set time when more than 30 seconds different
    DateTime now = rtc.now();
    double t = now.unixtime();
    double t2 = t - time_cs;
    t2 = abs(t2);
    if (t2 > 100 && time_cs != 0)
    {
      rtc.adjust(time_cs);
    }
    previousMillis8 = currentMillis8;
  }

  //CO2 control collections (pins 51,52 as serial connection)
  unsigned long currentMillis3 = millis(); // collection every 3 seconds
```

```
 if (currentMillis3 - previousMillis3 >= intervalco2)
  {
   //I2C comments below
   //co2 = K_30.getCO2('p');  //returns co2 value in ppm ('p') or percent ('%')
   co2 = readCO2();

   if (co2 <= 0)
   {
    //I2c comment
    //co2 = K_30.getCO2('p');} //returns co2 value in ppm ('p') or percent ('%')
    co2 = readCO2();
   }

   if (co2 <= 0)
   {
    //I2C comment
    //co2 = K_30.getCO2('p'); }//returns co2 value in ppm ('p') or percent ('%')
    co2 = readCO2();
   }

   if (co2 <= 0)
   {
    //I2C comment - this is left as is
    co2 = pco2;
   }

   co2_1.push(co2);
   co2_5.push(co2);
   duty_5.push(duty1);

   co2ave1= co2_1.mean(); // averages co2 1 min
   co2ave5= co2_5.mean(); // averages co2 5 min
   dutyave5=duty_5.mean(); // averages duty cycle 5 minute
   //co2sd5= co2_5.stddev(); // sd co25 minutes

   // resets if bad values for 1 minute; new for v3
   if(co2ave1 <= 0 && duty1 >= 425)
    software_Reset();

   float p10co2 =co2_1.get(3);
   float m = (co2-p10co2)/3; // slope based on 15 seconds previously ' changed from 5  to 10 in
v1.4
   float proj_co2= co2+3*m; // 15 second projection of c02 level
   float proj_dev = (treatment-proj_co2)/treatment;// 15 second projection of deviation
```

```
   duty1 = duty0 + (5 * proj_dev); // IMPORTANT change this constant for altering CO2 CHANGE
RESPONSIVENESS HIGHER NUMBER EQUAL MORE REACTIONS

   if (co2ave1 > 770)
     // IMPORTANT change this constant for altering CO2 CHANGE RESPONSIVENESS HIGHER
NUMBER EQUAL MORE REACTIONS
      duty1 = duty0 + (20 * proj_dev);

   //digitalWrite(Pin52, HIGH);2019 RLR change

   //if(dev<0){ duty1 = duty0 + (co2slope * dev);} ///old c02 code
   //if(dev>=00){ duty1 = duty0 + (2* co2slope * dev);}

   float pco2=co2;

   if (duty1 >= 500)
   {
    duty1=500;
   }

   if (duty1 <= 0)
   {
    duty1= duty0;
   }

   previousMillis3 = currentMillis3;
   dutyp = duty0; // previous duty cycle
   duty0 = duty1;
   //digitalWrite(Pin52, LOW); 2019 RLR Change
 }
 // end cycle

 unsigned long currentMillis6 = millis();
 if (currentMillis6 - previousMillis6 >= 10000)
 {
   /*reading of coil where i can succefully read value*/
   encodeFloat(&holdingRegs[aa], co2); // temperature variables to send
   encodeFloat(&holdingRegs[ba], duty1);
   encodeFloat(&holdingRegs[ca], gas);
   encodeFloat(&holdingRegs[da], co2ave5);
   encodeFloat(&holdingRegs[ea], dutyave5);
   //encodeFloat(&holdingRegs[fa], co2sd5);
   //encodeFloat(&holdingRegs[ga], Arduino_Code); // for trouble shooting
```

```
   encodeFloat(&holdingRegs[ka], temp);
   encodeFloat(&holdingRegs[la], humidity);
   encodeFloat(&holdingRegs[ia], pressure);

   dutycs = decodeFloat(&holdingRegs[ha]); //duty cycle overall control
   //dutySurface = decodeFloat(&holdingRegs[ia]); //duty cycle for surface heating
   CS_Code = decodeFloat(&holdingRegs[ja]); //control code fromcs
   //pinTemp_avg =  decodeFloat(&holdingRegs[ka]); //Ambient for comparison with other
plots heating operations
   //surfaceTemp_avg = decodeFloat(&holdingRegs[la]); //time from Campbell logger
   time_cs =  decodeFloat(&holdingRegs[ma]); //Ambient comparison with other plots check on
heating operations

   previousMillis6 = currentMillis6;
  }

  modbus_update(); //TODO does this need to go after collecting all vals or can it be before BME

  //if (! bme.performReading())
  //{
   //Serial.println("Failed to perform BMP reading :(");
   //return;
  //}

  // BME results
  unsigned long currentMillis7 = millis();
  if (currentMillis7- previousMillis7 >= 10000)
  {
   if (! bme.performReading())
   {
    //Serial.println("Failed to perform BMP reading :(");
    //return;
    temp = -7999;
    pressure = -7999;
    humidity = -7999;
    gas = -7999;
    approx_alt = -7999;
   }
   //else, here are real BME readings
   else
   {
    temp = bme.temperature; //*C
    pressure = bme.pressure / 100.0; //mbar
    humidity = bme.humidity; //%
```

```
    gas = bme.gas_resistance / 1000.0; //KOhms
    approx_alt = bme.readAltitude(SEALEVELPRESSURE_HPA); //m
   }

   previousMillis7 = currentMillis7;
  }

  // Revised LCD print screen
 // unsigned long currentMillis11 = millis();
 // if (currentMillis11 - previousMillis11 <= 10001)

  //{
   unsigned long currentMillis12 = millis();
   {

   if (currentMillis12 - previousMillis12 <= 5000)
   {
    lcd_state = 0;
   }
   if (currentMillis12 - previousMillis12 > 5000)
   {
    lcd_state = 1;
   }
   if (currentMillis12 - previousMillis12 >= 10000)
   {
    lcd_state2 = 0;
    lcd_state = 0;
    previousMillis12 = currentMillis12;
   }
   }
   if (lcd_state == 0 and lcd_state2 == 0)
   {
    lcd.clear();
    DateTime now = rtc.now();
    hr = now.hour();
    lcd.setCursor (0, 0);
    lcd.print("CO2 ppm = ");
    lcd.println(co2); //print value
    lcd.setCursor (0, 1); // go to the next line
    if (co2 <= 1000 && co2>= 0)
    {
     lcd.print ("CO2   OK");
    }
    if (co2 <= 0)
```

```
  {
    //co2=pco2;
    lcd.print ("CO2  rep");
  }
  if (co2 >= 10000)
  {
    co2=10000;
    lcd.print ("CO2 high");
  }
  lcd.setCursor (0, 2); // go to the next line
  lcd.print("duty = ");
  lcd.print(duty1);
  lcd.print (" ");
  lcd.print(dutyp); // print value
  lcd.setCursor (0, 3); // go to the next line
  lcd.print(now.year(), DEC);
  lcd.print('/');
  lcd.print(now.month(), DEC);
  lcd.print('/');
  lcd.print(now.day(), DEC);
  lcd.print(' ');
  lcd.print(now.hour(), DEC);
  lcd.print(':');
  lcd.print(now.minute(), DEC);
  lcd.print(':');
  lcd.print(now.second(), DEC);
  lcd_state2 = 1;
}

if (lcd_state == 1 and lcd_state2 == 1)
{
  lcd.clear();
  lcd.setCursor (0, 0);       // go to the next line
  lcd.print("temp = ");
  lcd.print(temp);
  lcd.setCursor(0,1); //line 2
  lcd.print("press = ");
  lcd.print(pressure);
  lcd.setCursor(0,2); //line 2
  lcd.print("humidity = ");
  lcd.print(humidity);
  lcd.setCursor(0,3); //line 2
  lcd.print("gas = ");
  lcd.print(gas);
```

```
   //altitude is going to be wrong unless we can accurately set sea level
   //lcd.setCursor(0,2); //line 3
   //lcd.print("approx alt = "+approx_alt+" ");
   lcd_state2 = 2;
  }


/* Display the current data set
  for (int i = 0; i < 30; i++)
  {
    Serial.print(co2ave.get(i));
    Serial.println();
  }
  for (int i = 0; i < 30; i++)
  {
    Serial.print(dutyave.get(i));
    Serial.println();
  }
*/
// And show some interesting results.
// Serial.print("Mean:   "); Serial.println(co2ave5.mean());
// Serial.print("Mean:   "); Serial.println(dutyave5.mean());
// Serial.print("Mode:   "); Serial.println(co2ave.mode());
// Serial.print("Max:    "); Serial.println(co2ave.maximum());
// Serial.print("Min:    "); Serial.println(co2ave.minimum());
// Serial.print("StdDev: "); Serial.println(co2ave5.stddev());
//
// duty cycle loop for controlling valve

  unsigned long currentMillis1 = millis();
  unsigned long startT1 = interval1 - duty1;

  if (currentMillis1- previousMillis1 <= startT1 )
  {
    State1 = LOW;
  }
  else
  {
    State1= HIGH;

  if (duty1 >= 499)
      State1 = HIGH;
    // time if then now
    //if (hr> 19) // new code for date time control
```

```
      //State1 = LOW;
     //if (hr< 6)
      //State1 = LOW; //new code for date time control

//
     if (hr> 19) // new code for date time control
      State1 = LOW;
     if (hr< 6)
      State1 = LOW; //new code for date time control
     if (CS_Code == 1)
      State1=LOW;  // all off
     if (CS_Code == 0)
      State1=State1;  // all normal operation
     }

   if (currentMillis1 - previousMillis1 > interval1) /// 2019 rlr CHANGE
     {
//       State1=LOW;
      previousMillis1 = currentMillis1;
     }

  digitalWrite(Pin34, State1); // may need to switch this outside of brackets

  //change to hour when logging data
  unsigned long currentMillis5 = millis();
  if (currentMillis5 - previousMillis5 >= 300000) // CHANGE TO 5 MINUTE 5-2019 rlr
  {
   DateTime now = rtc.now();
   digitalWrite(Pin53,HIGH);
   double t = now.unixtime();
//
//   Serial.print(now.year(), DEC);
//   Serial.print('/');
//   Serial.print(now.month(), DEC);
//   Serial.print('/');
//   Serial.print(now.day(), DEC);
//   Serial.print(' ');
//   Serial.print(now.hour(), DEC);
//   Serial.print(':');
//   Serial.print(now.minute(), DEC);
//   Serial.print(':');
//   Serial.print(now.second(), DEC);
//   Serial.println();
```

```
  // make a string for assembling the data to log:
  dataFile.print(t);
  dataFile.print(",");
  dataFile.print(now.year(), DEC);
  dataFile.print('/');
  dataFile.print(now.month(), DEC);
  dataFile.print('/');
  dataFile.print(now.day(), DEC);
  dataFile.print(' ');
  dataFile.print(now.hour(), DEC);
  dataFile.print(':');
  dataFile.print(now.minute(), DEC);
  dataFile.print(':');
  dataFile.print(now.second(), DEC);
  dataFile.print(",");
  dataFile.print(sl_id); //Slave ID
  dataFile.print(",");
  dataFile.print(co2);
  dataFile.print(",");
  dataFile.print(duty1);
  dataFile.print(",");
  dataFile.print(co2ave1);
  dataFile.print(",");
  dataFile.print(co2ave5);
  dataFile.print(",");
  dataFile.print(dutyave5);
  dataFile.print(",");
  dataFile.print("NA");
  dataFile.print(",");
  // BME added here
  dataFile.print(temp);
  dataFile.print(",");
  dataFile.print(pressure);
  dataFile.print(",");
  dataFile.print(humidity);
  dataFile.print(",");
  dataFile.print(gas);
  dataFile.print(",");
  dataFile.println(approx_alt); //make sure calibrated
  // flush() saves file to SD card after each line
  // slower, but safer. saves only every 512 bytes without flush().
  dataFile.flush();

//   // print to the serial port too:
```

```
//    Serial.print(t);
//    Serial.print(",");
//    Serial.print(sl_id);
//    Serial.print(",");
//    Serial.print(co2);
//    Serial.print(",");
//    Serial.print(duty1);
//    Serial.print(",");
//    Serial.print(co2ave1);
//    Serial.print(",");
//    Serial.print(co2ave5);
//    Serial.print(",");
//    Serial.print(dutyave5);
//    Serial.print(",");
//    Serial.print(co2sd5);
//    Serial.print(",");
//    //BME 680 added here
//    Serial.print(temp);
//    Serial.print(",");
//    Serial.print(pressure);
//    Serial.print(",");
//    Serial.print(humidity);
//    Serial.print(",");
//    Serial.print(gas);
//    Serial.print(",");
//    Serial.println(approx_alt); //make sure calibrated
//
    previousMillis5 = currentMillis5;
  }
}
```