Accelerated Wear - Final Design Review Report

Sponsor: Rob Brewster

By: Daniel Alanis (dalanis@calpoly.edu)

Tristan Frisella (tfrisell@calpoly.edu)

Kian Ashoubi (kashoubi@calpoly.edu)

Table of Contents

1.0 Executive Summary

Carbon dioxide accounts for a large amount of global greenhouse gases released into the environment. Because plastic degradation emits carbon dioxide, there has been interest to investigate how much carbon dioxide various plastics generate as they degrade. We have been tasked to create a dehumidifier and carbon dioxide sensor assembly for an accelerated wear testing apparatus that degrades plastic. Several products and literature currently exist that share similarity with our assembly but have not been manufactured to achieve the same combination of measurements we are expecting of our accelerated wear testing device. The following report shall introduce these products and literatures, highlight customer requirements, detail the design process, and discuss project management for our accelerated wear testing apparatus.

It has been shown with 95% confidence that our sensors output accurate results for CO_2, relative humidity, and temperature for the range of values that are expected to be encountered during the course of accelerated age testing of plastics inside a bioreactor. Additionally, it has been confirmed that the dehumidifying chamber can be used to dry a gas sample to a level that is appropriate for the sensors being used within this device.

Next steps for this device include, interfacing with a bioreactor, automation of sampling, a continuous method of drying the desiccant within the dehumidification chamber, along with machined parts to replace rapid prototyped components.

2.0 Introduction and Background

Introduction

According to the U.S. Environmental Protection Agency, carbon dioxide (CO_2) accounted for 65 percent of global greenhouse gases released into the environment in 2017 (1). Because carbon dioxide accounts for the largest percentage of greenhouse gasses emitted into the atmosphere, there have been multiple attempts to reduce CO_2 pollution such as the Clean Air Act of 1970 and incentivizing electric vehicle purchases.

Although there have been several measures to reduce CO_2 pollution, plastic waste remains a contributor of CO_2 emissions. Currently there are several patents such as patent numbers US9267875B2 and US4874952A that provide methods to degrade polymers at an accelerated rate, yet they do not quantify what by-products (such as CO_2) are emitted to the environment. Thus, there is still room for improvement in reducing CO_2 pollution and there has been interest to quantify the amount of CO_2 released by plastic waste.

We have been tasked by Rob Brewster and California Polytechnic State University, San Luis Obispo to investigate and create an accelerated wear testing apparatus which accelerates the degradation of plastic waste and measures the amount of CO_2 produced. This shall be accomplished by manufacturing a dehumidifier and obtaining a CO_2 transducer. The following work will discuss the background, objectives, and management with regards to our project.

Background

Table I: Existing designs

Design	Description
CO_2 sensor from CO2Meter.com	Detects CO_2 levels at an optimal sample rate of 500 mL/min for various ranges of CO_2 values
SmartBee - CO2 / LTH - Combined CO_2 & LTH Sensor (SB100102)	Designed for constant monitoring of light, temperature, humidity, and carbon dioxide levels
Titan Controls 702852 Saturn 6 Digital Environmental Controller with C	Provides the control of your temperature, humidity, and carbon dioxide PPM; levels with digital accuracy
QA Supplies 900101 cat oxygen (21%) and carbon dioxide (20%) analyzer	CAT portable gas analyzers are used to measure Oxygen and Carbon Dioxide levels inside a product shipping container

Table II: Related Patents

Patent Number	Description of Patent/Invention
US4874952A Granted 10/17/1989	Involves the accelerated photo-aging of materials containing polymers with a goal of better understanding photodegradation and determining the correlations between the life spans of said materials under accelerated photo-aging and climatic aging. Uses oxygen to quantify photodegradation.
US4957012A Granted 09/18/1990	Describes a method to predict the aging of polymers by heating a polymer in the outdoors to a high enough temperature where a change of property is induced. Uses solar radiation to quantify degradation.
US9267875B2 Granted 02/23/2016	Provides an accelerated life testing method for a test piece within a test chamber. This involves the establishment and alteration of multiple atmospheres within the chamber.
CN103351616A Granted 10/16/2013	Aims to solve the technical problem of providing a plastic anti-aging PA-PPS alloy with good anti-aging properties.
US20050004285A1 Granted 01/06/2005	Invention is a dimensionally-stable propylene polymer foam with improved thermal aging Uses stabilizing additives to quantify thermal degradation.

The apparatus we are creating has a high freedom to operate due to the lack of direct restrictions from any of the observed patents. Our product is not to be marketed as it is to be used for university research. Our apparatus will not infringe on any patents listed above.

Table III: Relevant Literature

Literature Title	Journal	Summary
Development of an automatic laboratory-scale respirometric system to measure polymer biodegradability	Polymer Testing 25, 2006 Kijchavengkul, Thitisilp	A respirometric system was built and tested to determine polymer degradation under simulated environmental conditions. Percentage of CO_2 produced was converted to percentage of mineralization. The system ran for 63 days and the PLA and PET plastics produced 64.2% and 2.7% mineralization.
Degradation process investigation of thermoplastic polyurethane elastomer in supercritical methanol	Polymer Degradation and Stability 98, 2013 Liu, Lu	Depolymerization of polyurethane was investigated in sub and supercritical methanol. The degradation started with breaking the urethane group and the polymer was separated into a soft and hard segment. The main products produced in the supercritical region were dimethyl adipate and 4,40-methylene diphenyl carbamate.
Effect of test parameters on degradation on polyurethane elastomer for accelerated life testing	Polymer testing 40, 2014 Kim, Hansol	Degradation characteristics of polyurethane sliding against stainless steel was investigated. Testing lasted 234 hours and rate of height decrease ranged from 0.15-0.9 micrometers/kilometers.

Table III Cont'd

High frequency circular translation pin-on-disk method for accelerated wear testing of ultrahigh molecular weight polyethylene as a bearing material in total hip arthroplasty	Journal of Biomechanics 48, 2015 Saikko, Vesa	Polyethylene was put under accelerated wear through a three-station, dual motion high frequency circular translation pin-on-disk was designed with a frequency of 25.3 Hz. In a 10-day test, the wear rate was 1.8 milligrams per day.
Plastic Degradation and Its Environmental Implications with Special reference to Poly(ethylene terephthalate)	Polymers 5, 2013 Webb, Hayden	Plastic has been increasing in global consumption and its resistance to degradation is of increasing concern to the environment, specifically polyurethane. Plastic goes through 4 stages of degradation and can take 50 or more years for it to fully degrade.

Our team shall be abiding by the following standards and regulations to create the dehumidifier and CO_2 sensor:

- ISO 14855 - This ISO standard describes ultimate aerobic biodegradability of plastic materials under controlled composting conditions, and the method utilized is analysis of evolved carbon dioxide.
- ASTM D5338 - This ASTM describes a standard testing method for aerobic biodegradation of plastics under controlled conditions and temperatures.
- ASTM D5988 - This ASTM describes a standard testing method for aerobic biodegradation of plastics specifically in soil.

3.0 Customer Requirements and Design Specifications

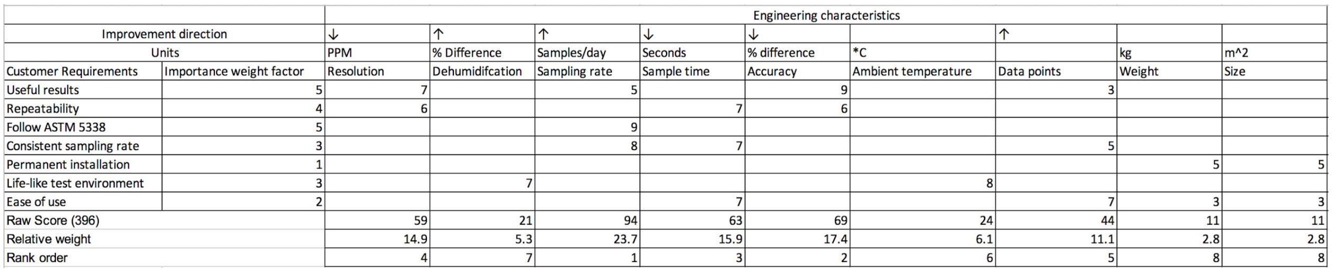
3.1 IFU

The Accelerated Wear Testing Reactor is intended for use with plastic material where measuring generation of CO_2 is desired. This testing reactor offers a selective technique of measuring CO_2 levels produced which is useful for evaluating long term environmental impact and sustainability.
It is intended for use in a standard industrial environment with ambient temperature where a cleanroom is not necessary.

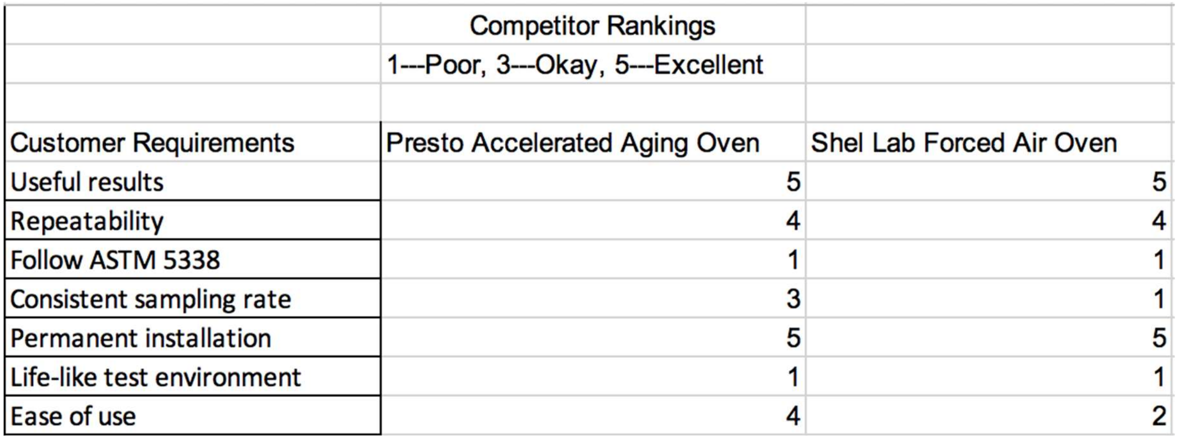
3.2 Product Design Specifications

Customer Requirement	Engineering metric	Specification	Reasoning
Must accurately measure CO_2 concentration	PPM of CO_2	Accurate to ±1000 ppm	This gives an accuracy of ±0.1%
Must accurately measure CO_2 concentration	Resolution (CO_2)	100 ppm	This allows for detection changes of 0.01% CO_2
Must give data on sample humidity	Absolute humidity	Accurate within 5%	Relative humidity is a function of temperature, absolute humidity is temperature independent. This specification is more lax than other specifications because the humidity data is not necessary for any of the standards set for this product.
Sample gas must be compatible with sensors	Relative Humidity	Dehumidifies to <80% RH	Some CO2 sensors have limits to the humidity they can function in. This specification is above the requirements of our sensors but has been included in case later iterations of the design use more sensitive sensors.
Sample gas must be compatible with sensors	Sample Flow Rate	>0.2 L/min	The minimum flow rate required to produce accurate results with the sensors.
Sample gas must be compatible with sensors	Volume of Sample Air	Sample 90% of reactor chamber volume	The internal volume of the sensing/dehumidifying chamber should be less than the sample chamber volume.

3.3 House of Quality



		Engineering characteristics								
Improvement direction		↓	↑	↑	↓	↓		↑		
Units		PPM	% Difference	Samples/day	Seconds	% difference	*C		kg	m^2
Customer Requirements	Importance weight factor	Resolution	Dehumidifcation	Sampling rate	Sample time	Accuracy	Ambient temperature	Data points	Weight	Size
Useful results	5	7		5		9		3		
Repeatability	4	6			7	6				
Follow ASTM 5338	5			9						
Consistent sampling rate	3			8	7			5		
Permanent installation	1								5	5
Life-like test environment	3		7				8			
Ease of use	2				7			7	3	3
Raw Score (396)		59	21	94	63	69	24	44	11	11
Relative weight		14.9	5.3	23.7	15.9	17.4	6.1	11.1	2.8	2.8
Rank order		4	7	1	3	2	6	5	8	8



	Competitor Rankings	
	1---Poor, 3---Okay, 5---Excellent	
Customer Requirements	Presto Accelerated Aging Oven	Shel Lab Forced Air Oven
Useful results	5	5
Repeatability	4	4
Follow ASTM 5338	1	1
Consistent sampling rate	3	1
Permanent installation	5	5
Life-like test environment	1	1
Ease of use	4	2

Figure 1: From left to right, respectively: The Presto Accelerated Aging Oven and the Shel Lab Oven

HQQ Findings

The house of quality's importance factor showcases the desire for this project's accelerated wear testing apparatus to useful results and repeatability while following ASTM guidelines. While the ovens are slightly different than the apparatus being built in this project, they still made for a decent comparison. This project's apparatus was found to be stronger in customer requirements such as test environment, consistent sampling rate, and following ASTM guidelines. The house of quality also ranked the importance of several engineering characteristics and found that the three most important for this project were sampling rate, accuracy, and sampling time.

4.0 Stage Gate Process

4.1 Concept Review

Concept Design 1

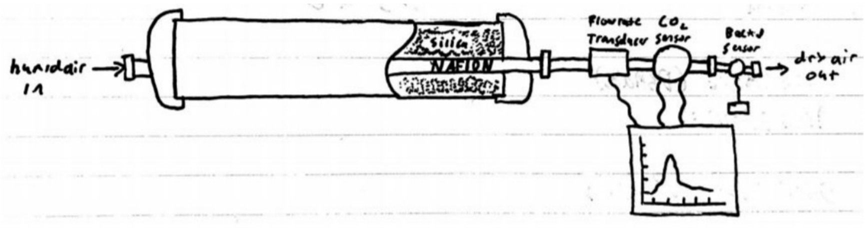


Figure 2: This concept design depicts the dehumidifying and testing apparatus for the accelerated wear testing device. Humid air from the reaction chamber enters the Nafion tubing which has been encased in PVC tubing, the drying tube has been filled with silica gel beads which will draw the moisture from the humid air before reaching the measuring devices.

Concept Design 2

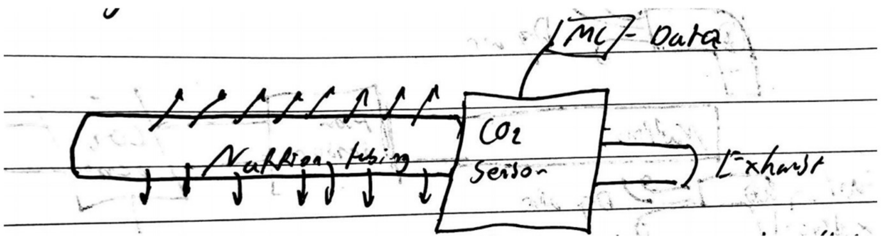


Figure 3: Our second concept design simplifies our apparatus and appears the least attractive. It depicts the Nafion tubing dryer leading into the carbon dioxide sensor, which is attached to the microcontroller and display for data analysis. Exhaust (air) is provided an exit on the right of the sensor.

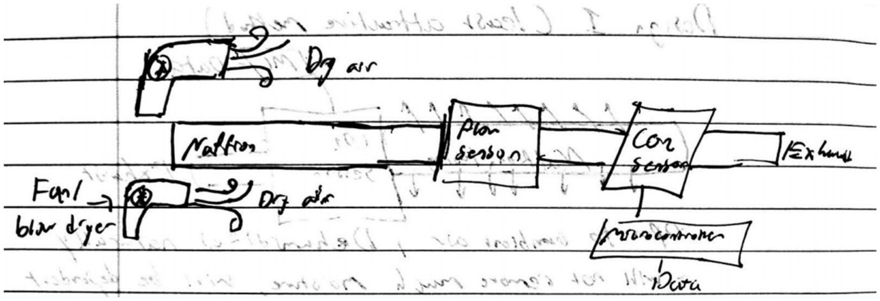


Figure 4: Our third concept design depicts the dehumidifying and testing apparatus for the accelerated wear device with dry air being blown on the surface of the apparatus. Humidified air shall enter the Nafion tubing and the dry air on the outside shall draw moisture outside the Nafion membrane before it reaches the measuring devices.

Front Runner Concept: Concept Design 1

The first concept design was selected as our front runner concept for our project because we deemed it our most likely to succeed. Our Pugh chart shows that this concept bested the Shel Lab Forced Air Oven in five categories, while losing to it in zero categories. Our second concept did not provide better humidity, and our third concept was inferior in output temperature due to the use of a blow dryer. Temperature at the end was one of our least important criteria, so we are content with matching the output temperature of the Shel Lab Forced Air Oven. We deemed the blow dryer design inferior and not as optimal as our design in concept one due to the inconvenience and labor that blow dryers provide.

4.2 Design Freeze

In the design freeze, preliminary prototype manufacturing and testing plans were explained. The preliminary design for the prototype of this project's apparatus was showcased using dimensioned CAD drawings along with material selections and cost estimations.

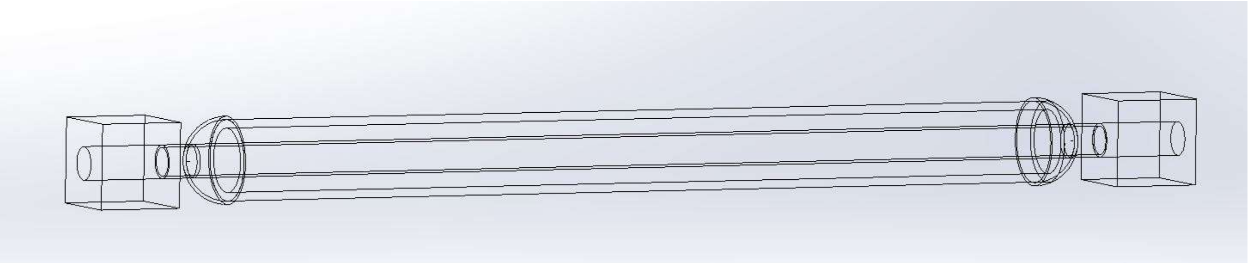


Figure 5: Preliminary prototype design

Figure 5 displays the preliminary prototype design, which includes a 1.5 inch by 10-foot PVC pipe and 1.5-inch PVC caps surrounding the 0.065 inch by 24 inch coiled Nafion tubing. Nafion, a sulfonated tetrafluoroethylene-based fluoropolymer-copolymer, was selected as a dehumidifying material for air samples to run to prior to analysis. Silica gel desiccant was selected to be filled into the tubing for further adsorption of water vapor from humid samples. The dehumidification chamber remained connected to the other components of the apparatus as shown in figure 2 in the concept design.

Initial manufacturing and testing methods were modified significantly for the final design. The initial manufacturing process included cutting and sanding the 10-foot PVC into 21 inches, but alterations were made in the design and manufacturing in the design review. Testing plans for carbon dioxide sensors testing initially included a gas chromatography machine, but alternative testing methods were used for the final design.

4.3 Design Review

The final design for this project included changes to the manufacturing and testing plan. For the manufacturing plan, a clear polycarbonate pipe was eventually decided upon for the dehumidifying chamber to ensure that users can identify when the color changing desiccant needs to be dried or replaced. As shown in figure 6, the dimensions of the polycarbonate pipe changed dramatically to 1.5 inch by 6 inches for the final design. This was due to the decision to coil the Nafion tubing around a 6.05 inch-long, 1.25-inch-wide 3D-printed support fitted and placed inside the pipe.

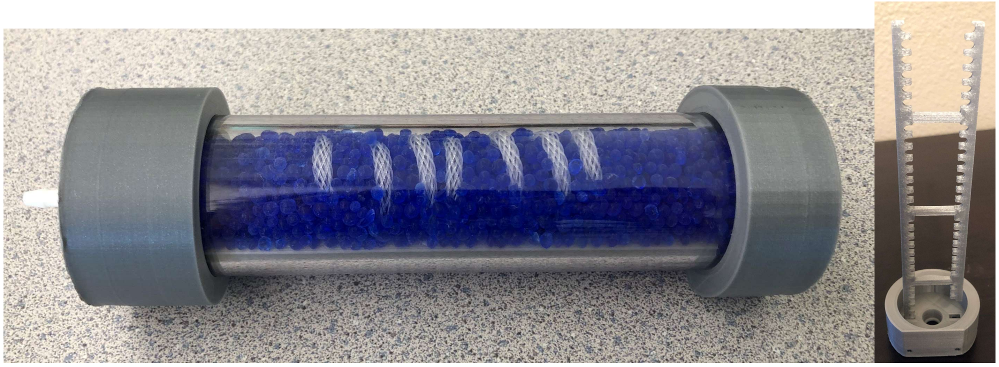


Figure 6: From left to right: final dehumidification chamber and 3D-printed support piece attached to one of the two 3D-printed caps

The final design included relative humidity sensors on either side of the dehumidification chamber but otherwise aligned with figure 2 of the concept review. For the final testing plan, carbon dioxide sensor testing was conducted with a CA-10 carbon dioxide analyzer from sable systems. Updates were made accordingly to the testing plan as well as the cost breakdown. Further details of the entire apparatus design can be found in section 5.0 of this report.

5.0 Description of Final Prototype Design

5.1 Overview

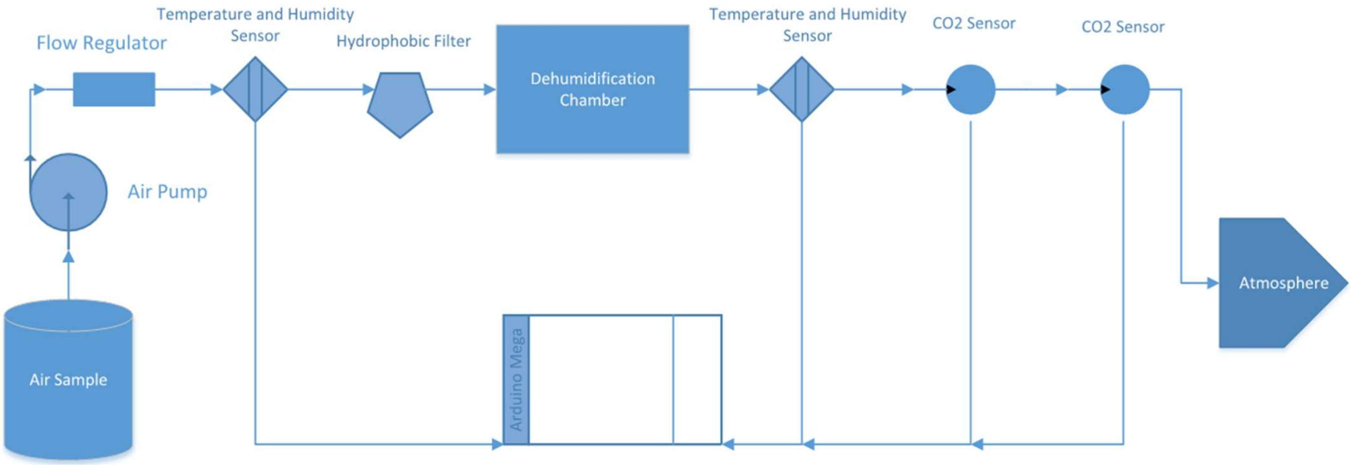


Figure 7: Air flow piping and instrumentation diagram for the dehumidification chamber and sensors

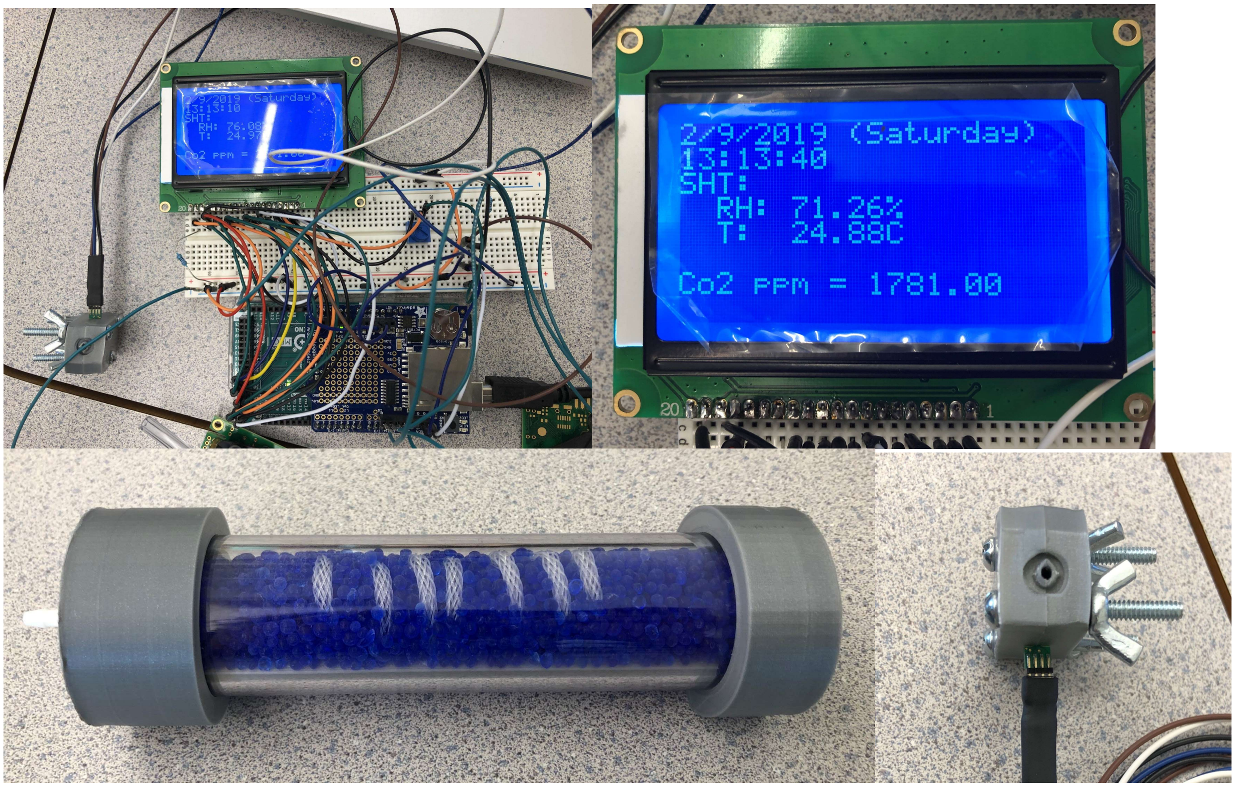


Figure 8: Physical components and sensors utilized for dehumidification and data acquisition

The overall diagram above demonstrates that we will obtain a flow regulated air sample, measure the initial temperature and humidity, pass the sample past a hydrophobic filter and through the dehumidification chamber, and record the new temperature and humidity values along with gathering two CO_2 values, and the sample is finally released into atmosphere. The sample data was recorded on an SD card for future reference.

5.2 Design Justification

The original design utilized a dehumidification chamber that was the same length as the Nafion tubing (24 in.). This design was updated for the final prototype where a shorter tube (6 in.) was used and the Nafion tubing coiled inside. This gave a 75% reduction in the length of the dehumidification chamber while giving the same effective dehumidification.
Multiples of each sensor were used in order to protect against error. Both sensors should output the same value, so if the results from one sensor does not match the other it will be evident that at least one of the sensors is inaccurate and steps can be taken to calibrate or replace a sensor.

5.3 Analysis

Our final design is in compliance with our product design specifications as shown in section 3.2. For instance, the CO_2 sensors acquired from CO2meter have been effective of being accurate within 1000 PPM of CO_2 when compared to a validated gas sample. The CO_2 sensors also have a resolution of 10 PPM, which is much greater than the 100 PPM that our customer requested. As for the relative humidity and temperature sensors, they have demonstrated their effectiveness as they are within 5% accuracy of absolute humidity when compared to a validated humidity sample. The dehumidification chamber has been shown to reduce the humidity of our air sample below 80% relative humidity during testing. Lastly, q sample flow rate greater than 0.2 L/min was achieved with a pump that outputs a flow rate at 1 L/min, thus we were assured that we were obtaining accurate CO_2 readings. Volume sample of air criteria listed on section 3.2 has been excluded from the final design as it was not within the scope of the project.

5.4 Cost Breakdown

Item Description	Manufacturer/ Distributor	Product Number	Purpose	Unit	Quantity	Cost/Unit	Total Cost
Nafion Tubing	CO2Meter	TUB-0003	Dehumidify	EA	1	$149.00	$149.00
Filters and Water Trap Kit	CO2Meter	CM-0103	Dehumidify	EA	1	$49.00	$49.00
Silica Gel	Amazon	B016VHQ2B6	Dehumidify	1 Qt.	1	$15.99	$15.99
10% CO_2 meter	CO2Meter	SE-0025	Sensing	EA	2	$249.00	$498
Temperature and Humidity Sensor	Sensiron	SHT85	Sensing	EA	4	$29.22	$116.88
Arduino Mega 2560 +3.2" TFT Display	SainSmart	101-52-C17	Data Logging	EA	1	$39.99	$39.99
SDMemory Card (8 GB SDHC)	Adafruit	1294	Data Logging	EA	1	$9.95	$9.95
Data Logging Shield	Adafruit	1141	Data Logging	EA	1	$13.95	$13.95
TCA9548A I2C Multiplexer	Adafruit	2717	Data Logging	EA	1	$6.95	$6.95
1.5"x6" clear polycarbonate	Rob Brewster	ZAM2966-B0070Z70WE	Housing	EA	1	$0.00	$0.00
Air Pump	Rob Brewster	HG10800	Testing	EA	1	$0.00	$0.00
						Total	**$899.71**

5.5 Safety Considerations

The dehumidifying and sensing apparatus is a low risk part but may radiate some heat due to the hot gasses passing through it. The apparatus will be housed in an insulated container which will serve the dual purpose of protecting the apparatus from ambient conditions and protecting the user from the radiated heat. As with any electronics there is a risk of electrocution; no component in this device requires more than 12 V so this risk is low. Electrical components have the potential to overheat if improperly connected.

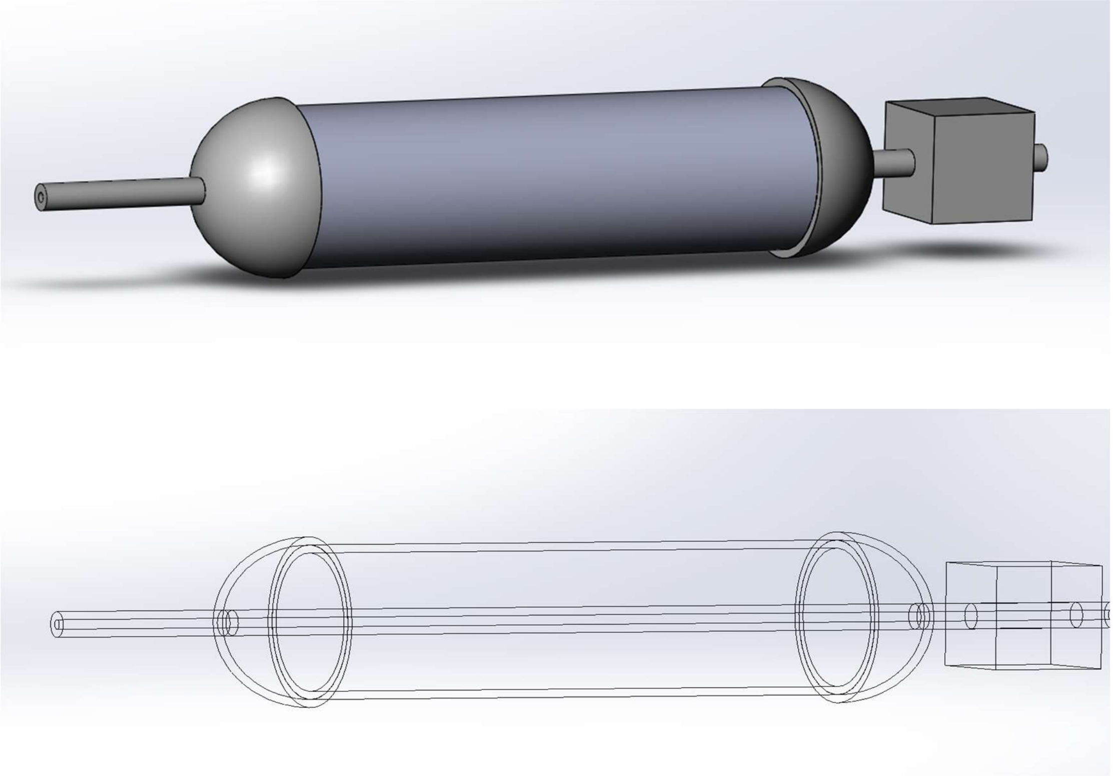
6.0 Prototype Development

6.1 Model Analyses

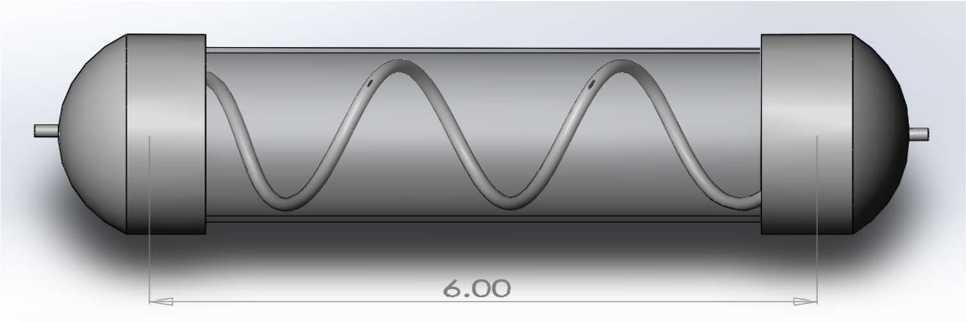
For our first design, we are interested in drying the desiccant that is responsible for dehumidifying the gas sample in preparation for measuring the CO_2 output for degrading plastics. This apparatus will interface with a 12-chamber bioreactor that will create a hot and humid environment in accordance with ASTM 5338 standards. The high humidity produced in the reaction chambers has the potential to damage many high accuracy CO_2 meters and as such it is often required to dry the gas sample prior to performing this analysis. The gas sample will be pushed through a dehumidifying chamber that will then be placed in line with various sensors in order to collect data about the degradation of plastics placed into the bioreactor (CO_2 level, relative humidity, temperature).

See SolidWorks models in the following section.

6.2 Evolution of Prototypes



Prototype 1 design: Nafion tubing sitting inside the dehumidifying chamber. The block represents the CO_2 and temperature sensors.

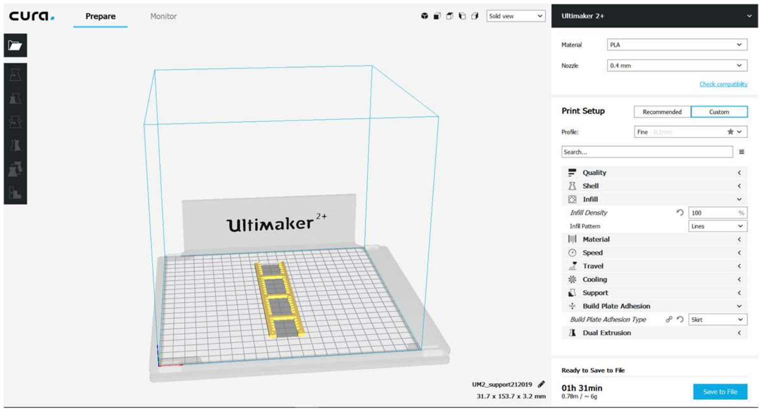


Prototype 2 design: Nafion tubing coiled around the dehumidifying chamber. This allows us to shorten the tube length yet have the same effective amount of Nafion tubing in the dehumidifying chamber.

6.3 Manufacturing Process

3D Printing

1) Upload the .STL file of the component for printing to the 3D printer
2) Ensure the component is appropriately oriented
3) Use the proper settings for your material and printer. All components are PLA, and components that require additional machining were made solid, the others were hollow.



Ultimaker 2+

Material PLA

Nozzle 0.4 mm

Print Setup Custom

Profile: Fine - 0.1mm

Infill Density 100 %

Infill Pattern Lines

Build Plate Adhesion Type Skirt

UM2_support212019

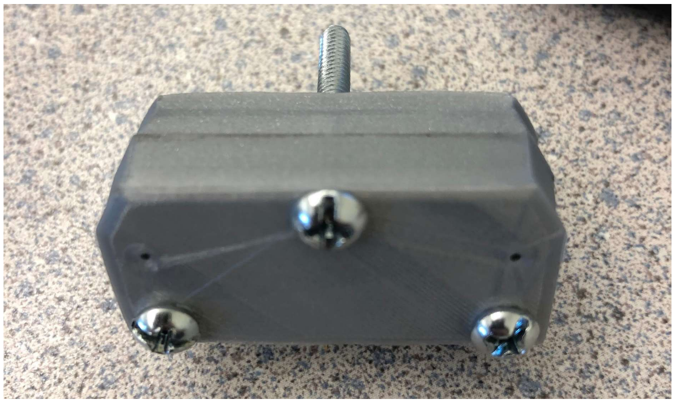
31.7 x 153.7 x 3.2 mm

Ready to Save to File

01h 31min

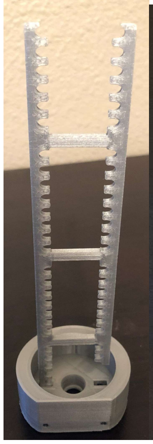
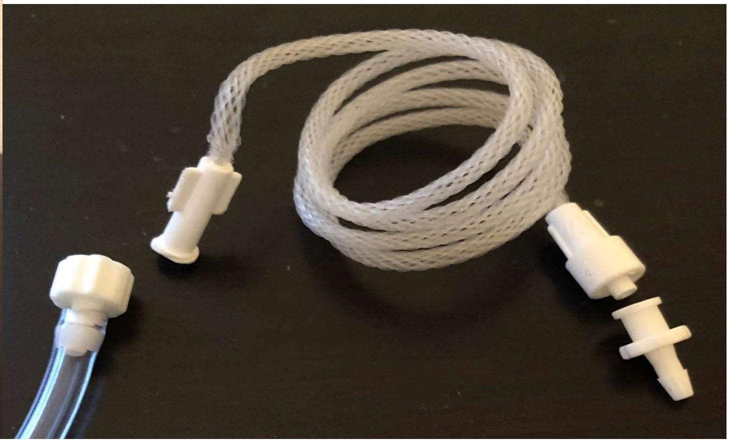
0.78m / ~ 6g

4) Print the component
5) Remove excess material and sand components until they have a firm connection
6) Drill through the pilot holes. 3/16" 6-32 screws were used holes were drilled with a 1/8" drill bit and the holes were left untapped.

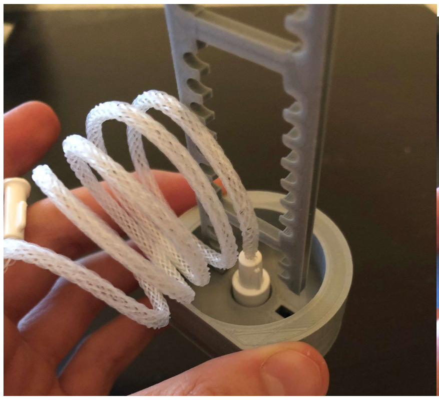
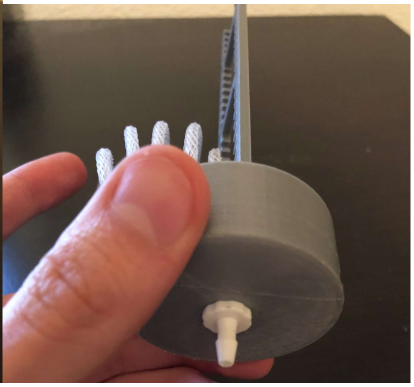


Dehumidifying Chamber

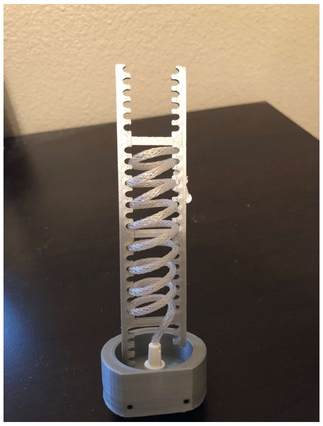
1) Gather Nafion tubing, polycarbonate piping, the 3d printed end caps with the support, silica gel desiccant, and Tygon tubing.

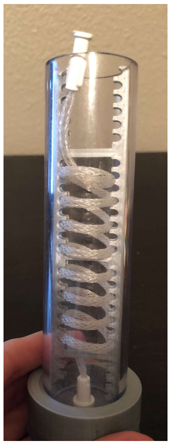
2) Screw the end pieces of the Nafion tubing together so that they lock around the end cap with the support

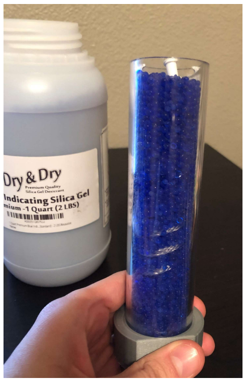
3) Coil the Nafion tubing around the support structure



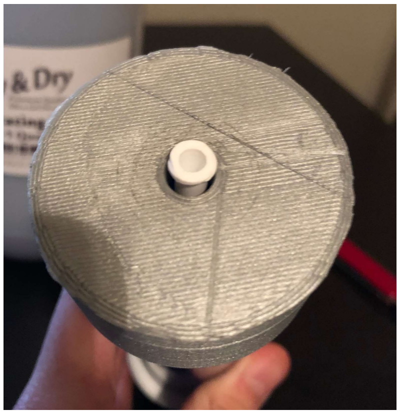
4) Insert the support into the polycarbonate pipe



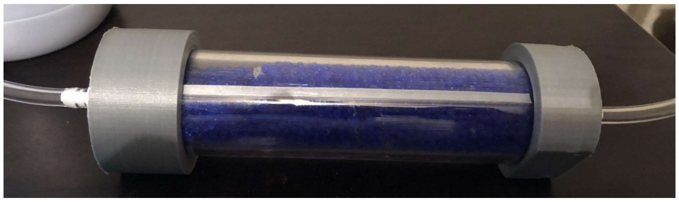
5) Fill the tube with the desiccant



6) Screw the end pieces of the Nafion tubing into the other end cap and connect the end cap

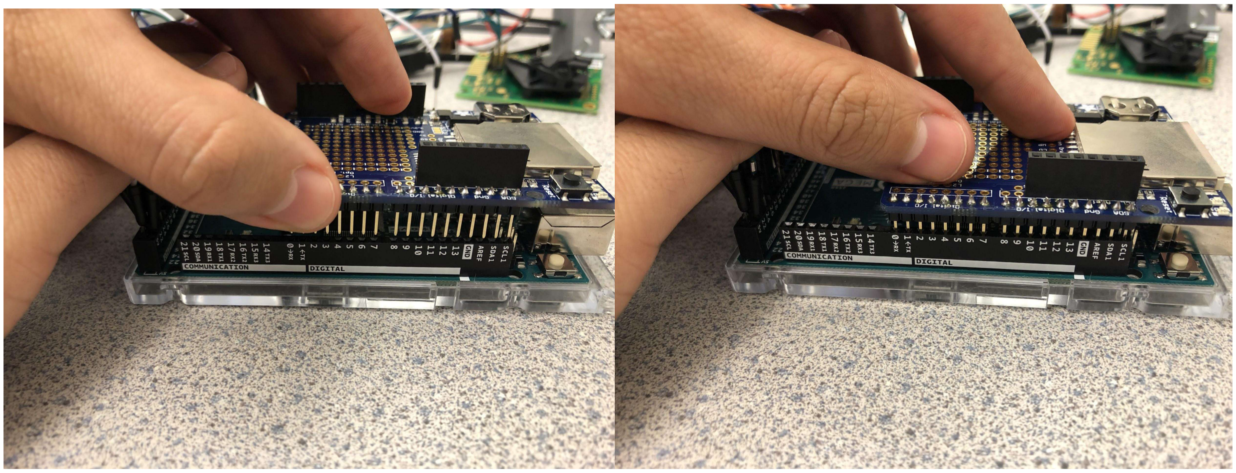


7) Cut two 3” lengths of Tygon tubing and attach them to the outputs at each end of the chamber



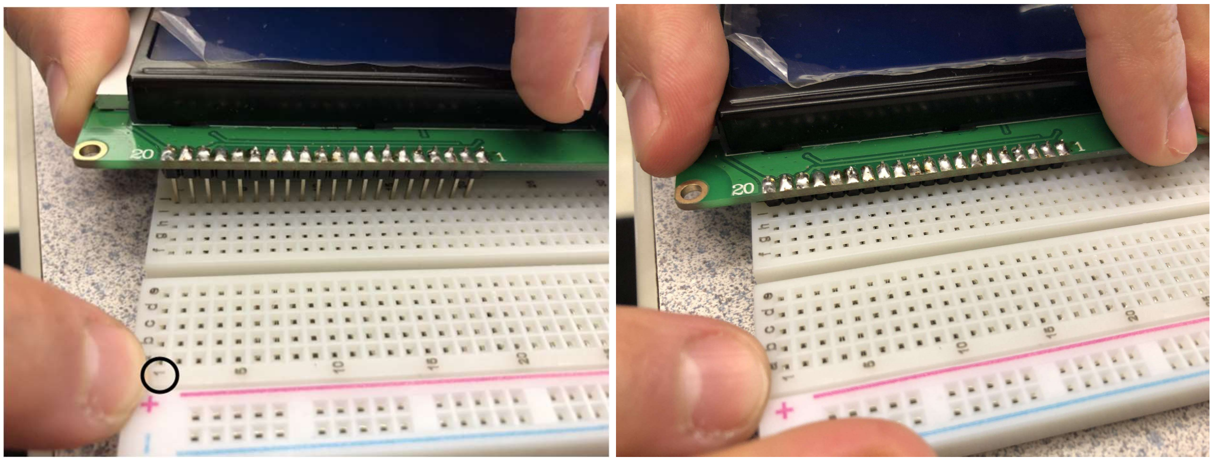
Data Logging Shield

1) Connect the data logging shield to the Arduino Mega as shown below

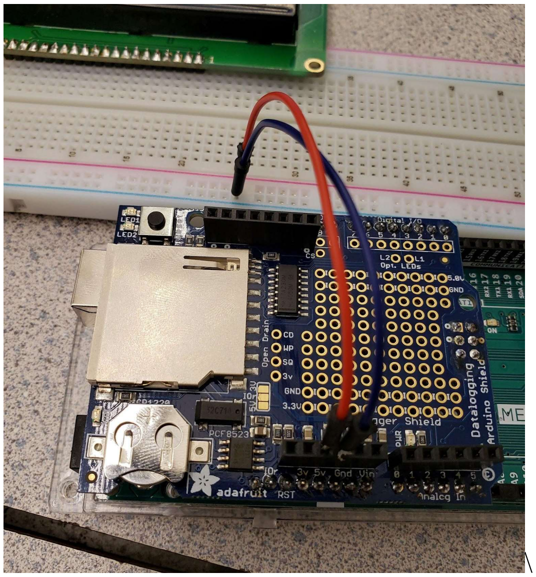


Screen

1) Connect screen to location 1 on breadboard as shown below

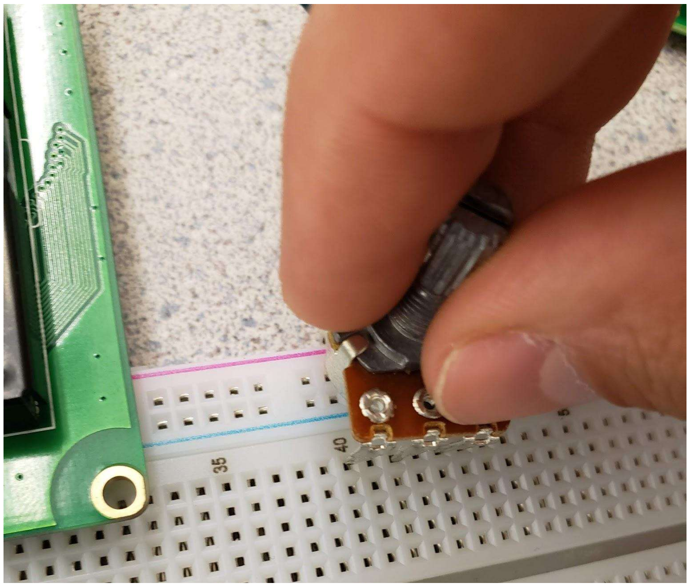


2) Connect 5V to red rail and ground to blue rail at location 25 on the breadboard as shown below



\

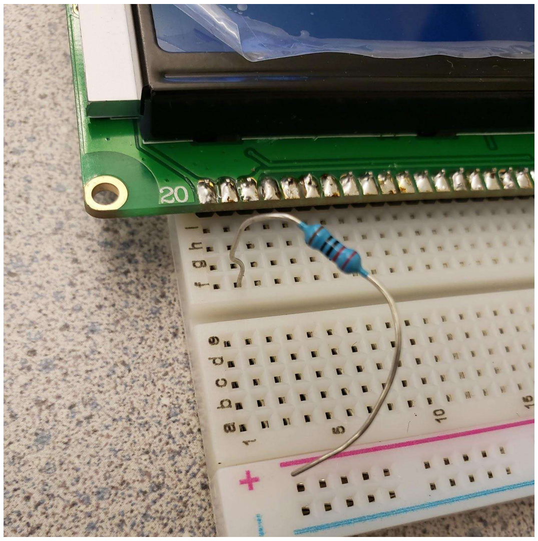
3) Connect potentiometer to location 40 on the breadboard as shown below and connect location 44 on the breadboard to the ground rail



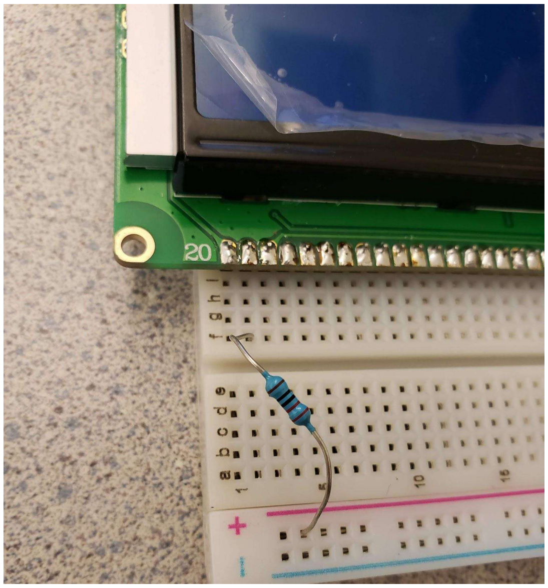
Note: Refer to figure 9 for visual aid.

4) Connect screen pin 1 VSS to ground rail
5) Connect screen pin 2 VDD to 5V rail
6) Connect screen pin 3 VO to 1S on the potentiometer (location 42 on breadboard)
7) Connect screen pin 4 RS to Arduino Mega pin 36
8) Connect screen pin 5 RW to Arduino Mega pin 35
9) Connect screen pin 6 E to Arduino Mega pin 37
10) Connect screen pin 7 DB0 to Arduino Mega pin 22
11) Connect screen pin 8 DB1 to Arduino Mega pin 23
12) Connect screen pin 9 DB2 to Arduino Mega pin 24
13) Connect screen pin 10 DB3 to Arduino Mega pin 25

14) Connect screen pin 11 DB4 to Arduino Mega pin 26
15) Connect screen pin 12 DB5 to Arduino Mega pin 27
16) Connect screen pin 13 DB6 to Arduino Mega pin 28
17) Connect screen pin 14 DB7 to Arduino Mega pin 29
18) Connect screen pin 15 CS0 to Arduino Mega pin 33
19) Connect screen pin 16 CS1 to Arduino Mega pin 34
20) Connect screen pin 17 /RST to 5V rail
21) Connect screen pin 18 VEE to 1E on the potentiometer (location 40 on breadboard)
22) Connect screen pin 19 A to one end of the 220-ohm resistor as shown below



23) Connect open end of 220-ohm resistor to 5V rail as shown below



24) Connect screen pin 20 K to ground rai
25) Turn the knob on the potentiometer to adjust contrast

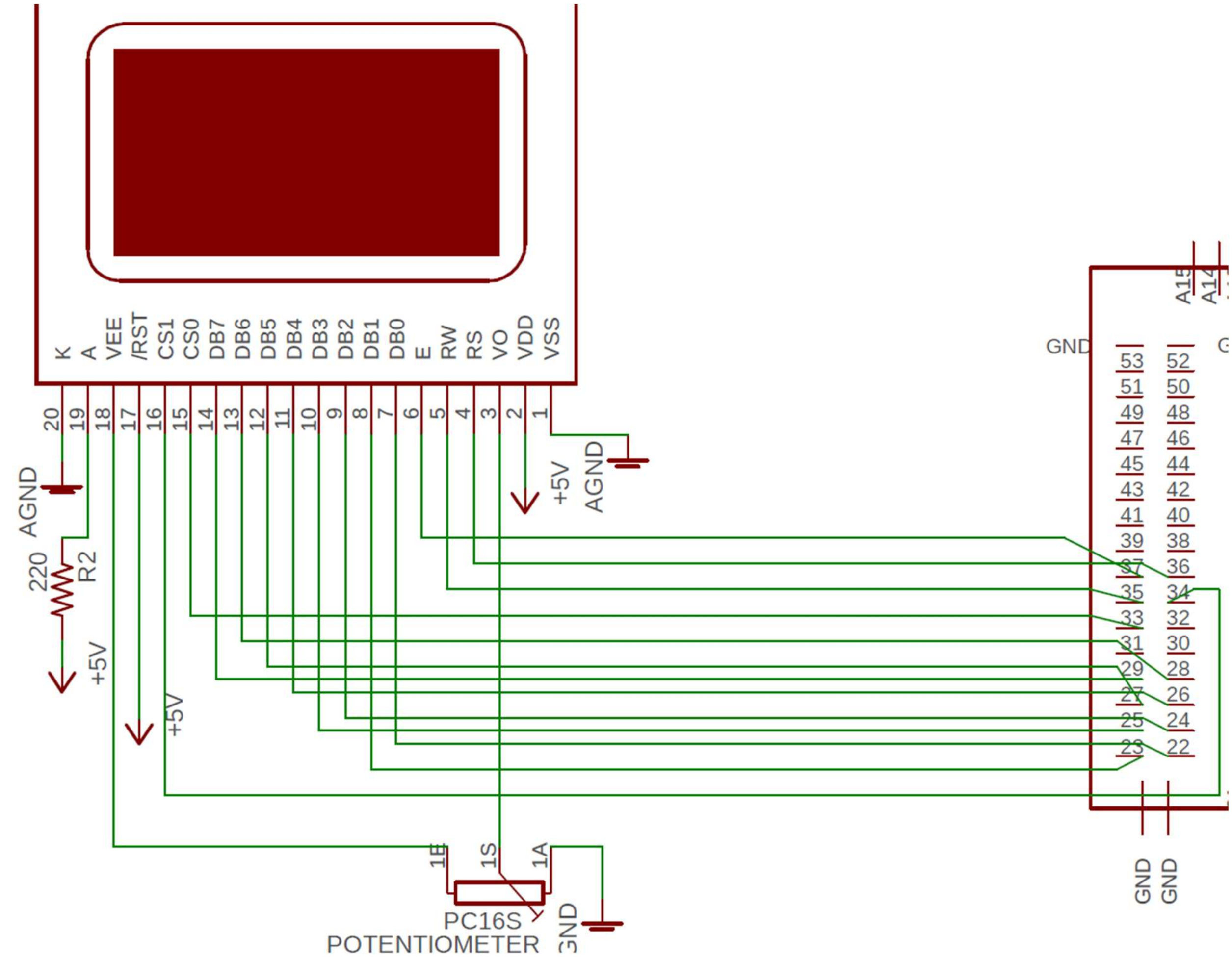
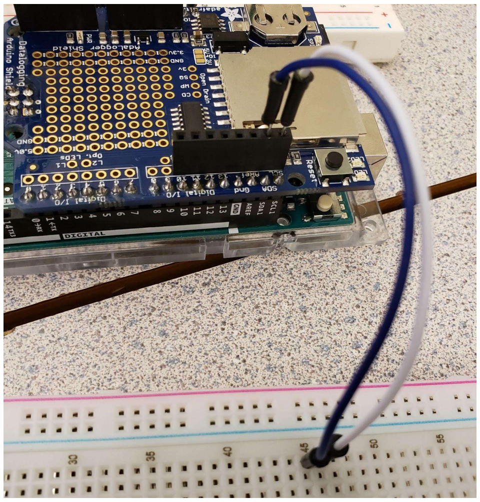


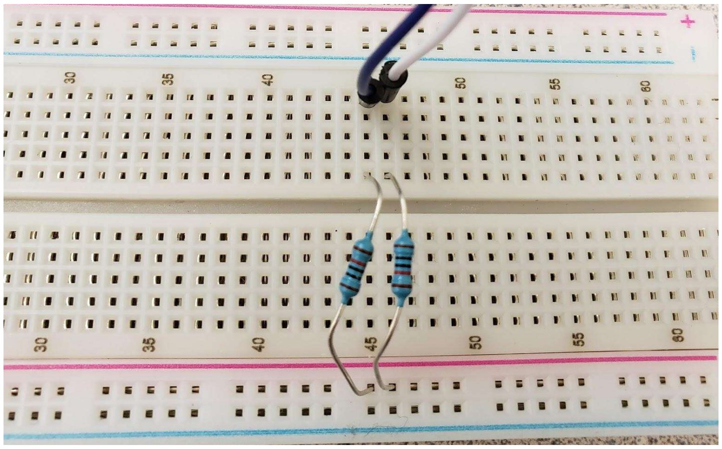
Figure 9: Wiring diagram for LCD assembly

Temperature and Humidity Sensor

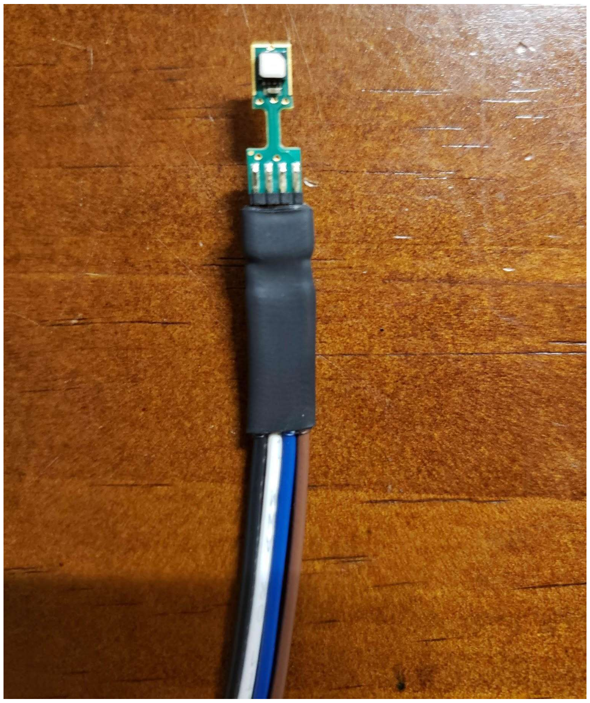
1) Connect SDA1 and SCL1 from the data logging shield to locations 45 and 46 respectively as shown below



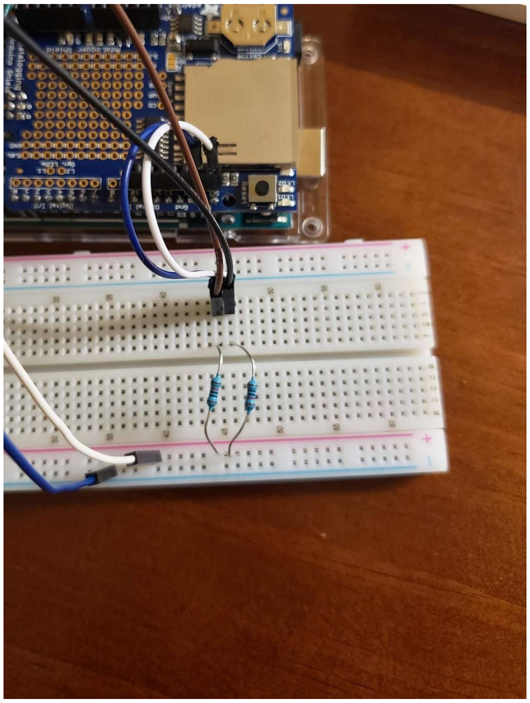
2) Connect a 10k resistor from the 5V rail to locations 45 and 46 as shown below



3) Using the adapter provided, connect the temperature and humidity sensor as shown below



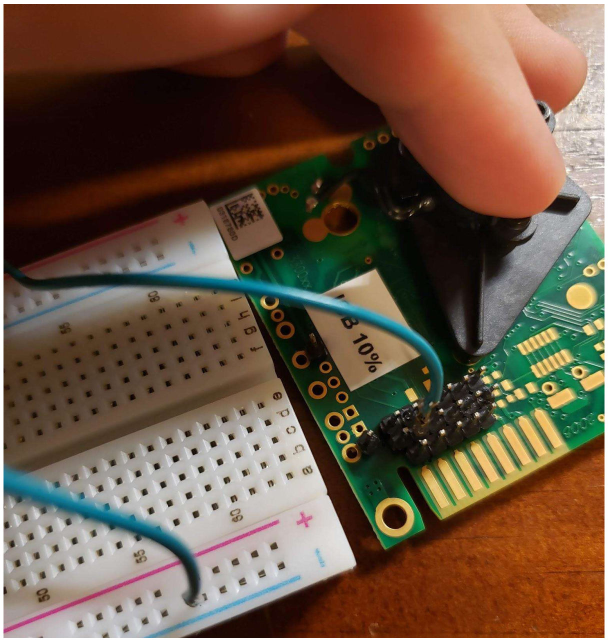
4) Connect the wires in the following order from left to right: Connect the black wire to SCL1 (location 46 on breadboard), the white wire to the 5V rail, the blue wire to the ground rail, and the brown wire to SDA1 (location 45 on breadboard) as shown below



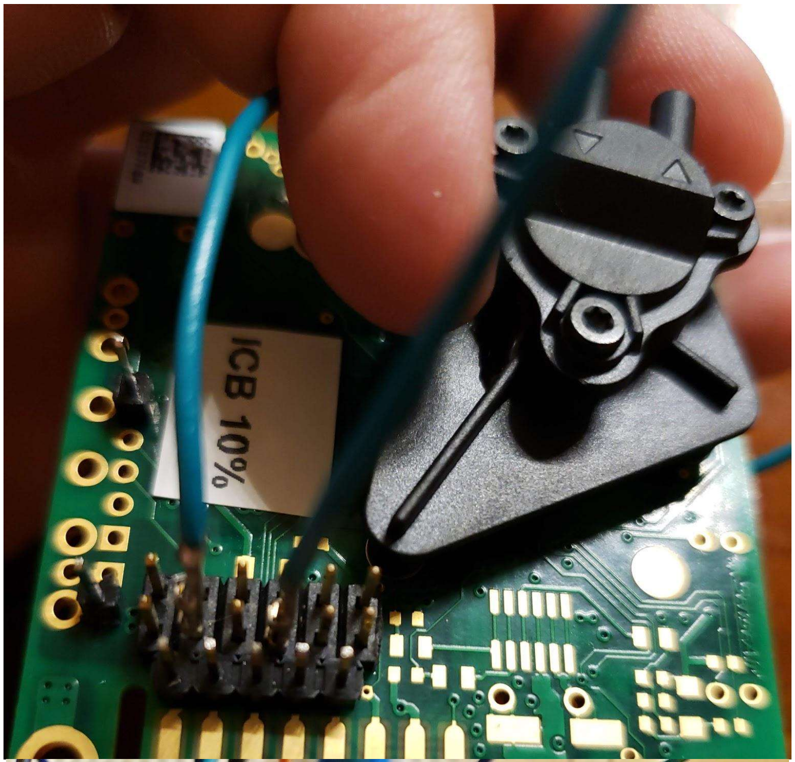
5) Repeat steps 3-4 once more for the second sensor

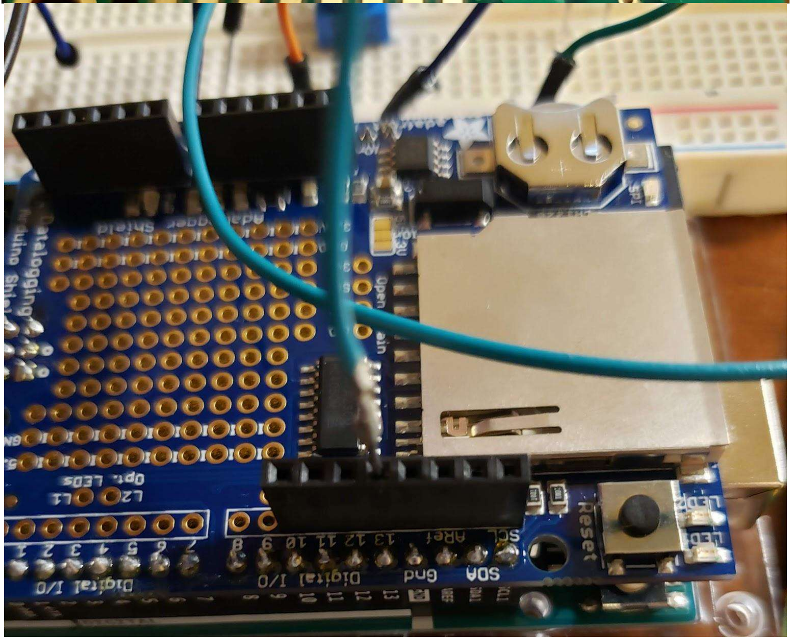
CO_2 Sensor

1) Connect the pin shown below to the ground rail as shown below

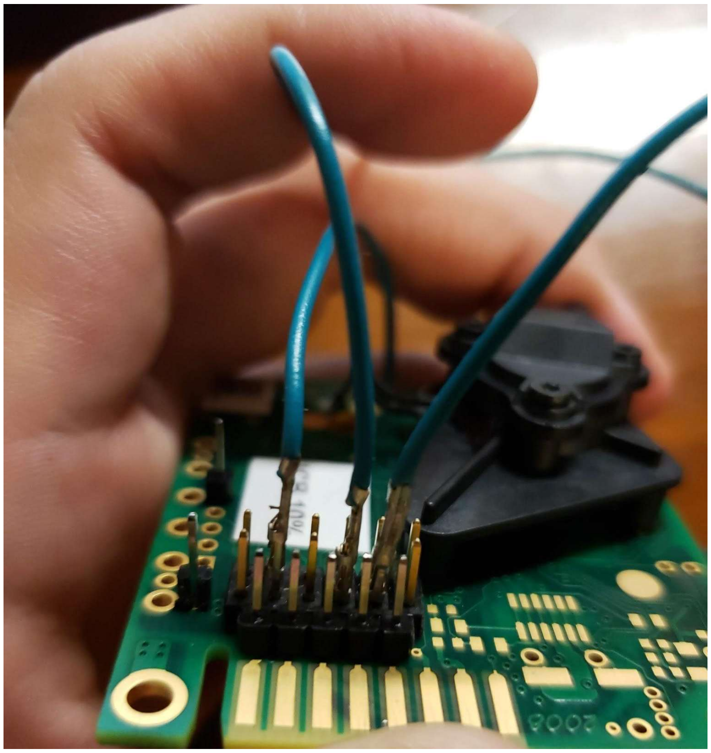


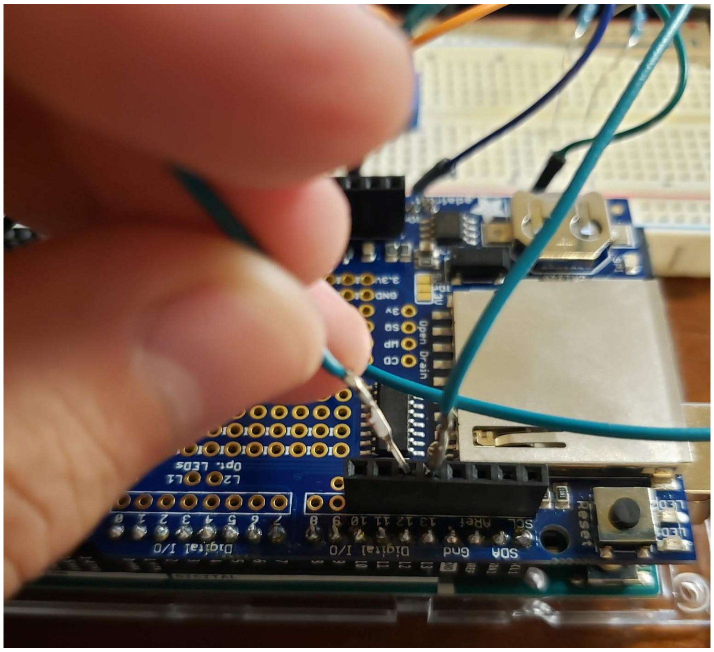
2) Connect the right pin shown below to pin 13 on the data logging shield



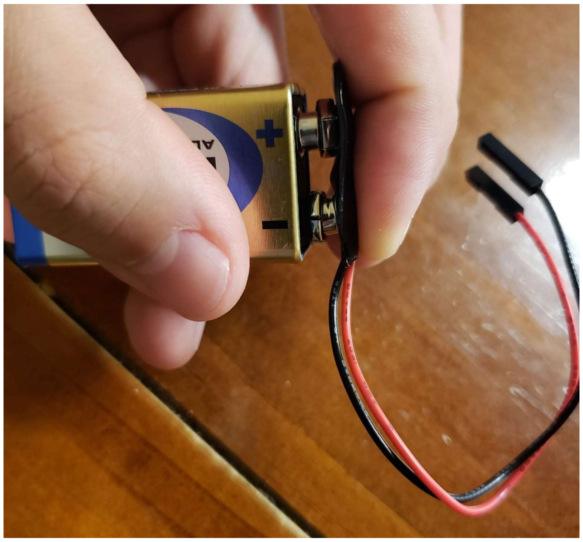


3) Connect the farthest right pin shown below to pin 12 on the data logging shield

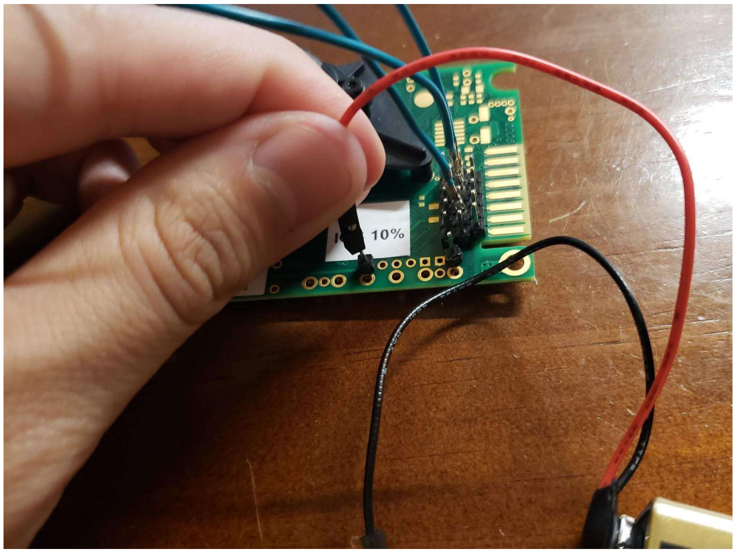




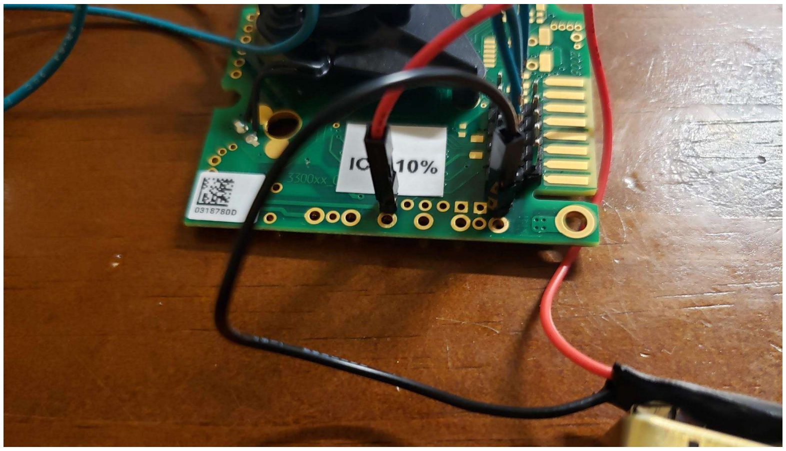
4) Connect the battery to the battery adapter as shown below



5) Connect the red wire to the left pin as shown below

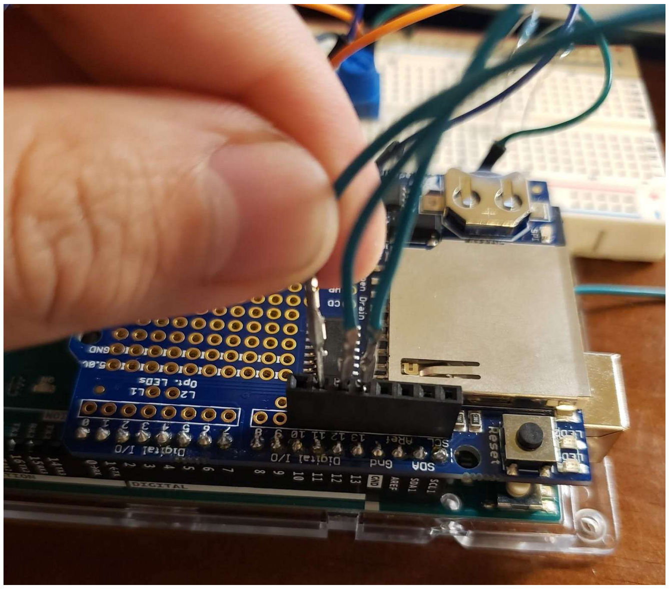


6) Connect the black wire to the right pin as shown below

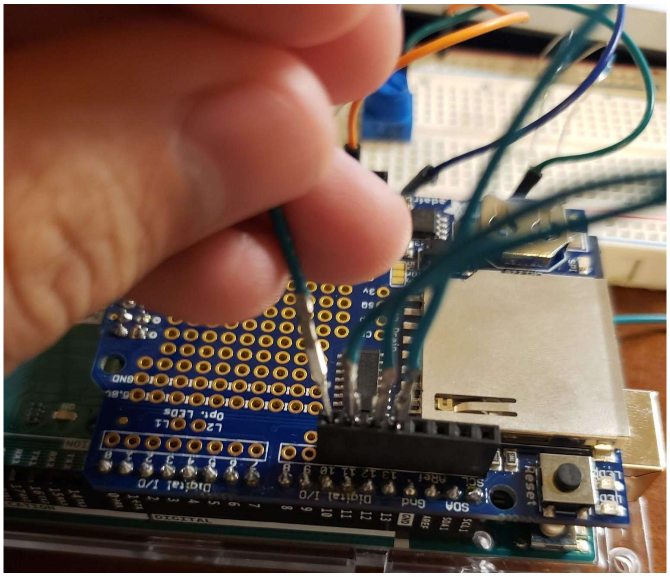


7) Repeat step 1 for the second CO_2 sensor

8) Repeat step 2 but connect the right pin to pin 11 on the data logging shield as shown below



9) Repeat step 3 but connect the farthest right pin to pin 10 on the data logging shield as shown below

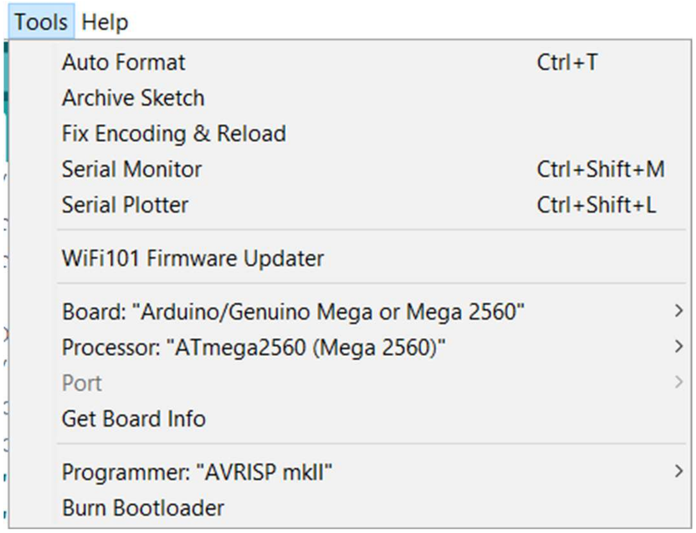


10) Repeat steps 4-6 for the second sensor

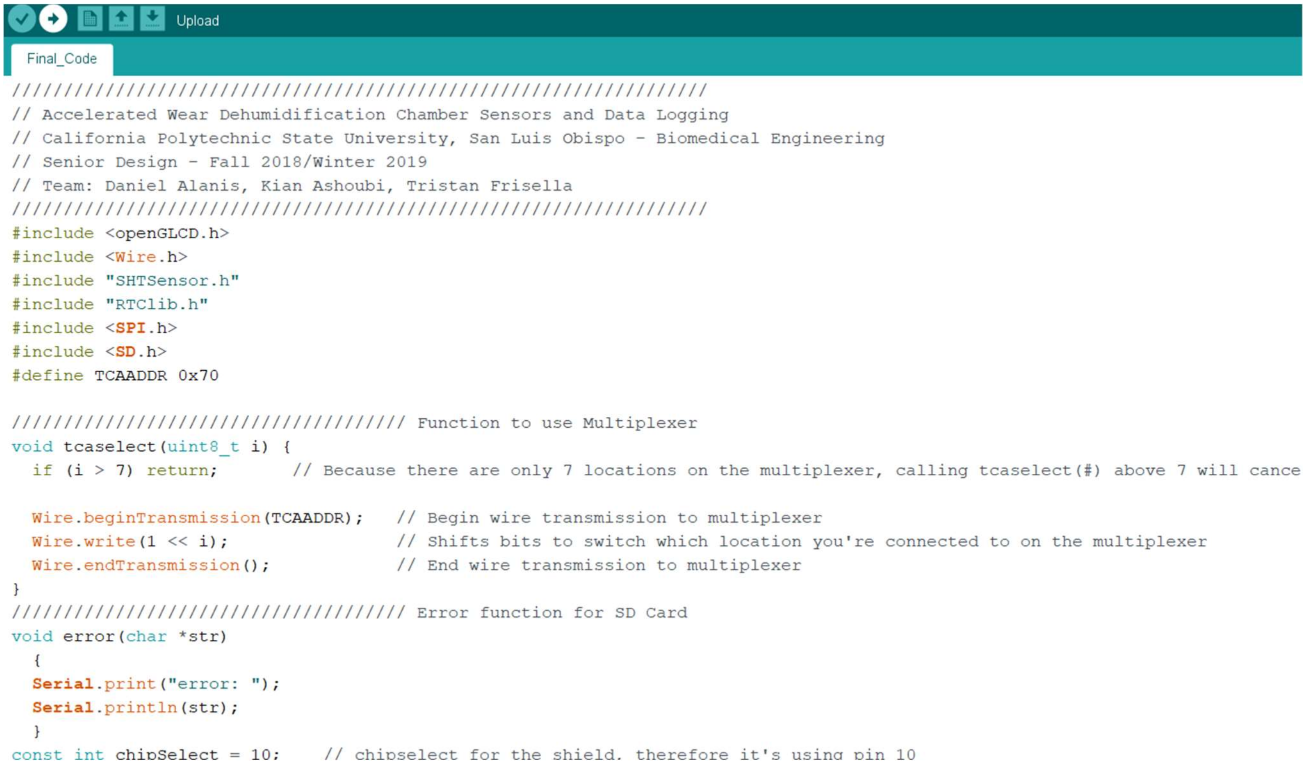
Uploading software to Arduino

1) Open the Arduino IDE software
2) Connect the Arduino Mega via USB to the computer
3) On the ‘Tools’ tab on the upper left-hand corner, select “Arduino/Genuino Mega or Mega 2560” under ‘board’ and select the appropriate ‘port’ for the Arduino Mega as shown below

   Note: Each computer sets a unique ‘port’ for the Arduino Mega. Ensure only 1 Arduino Mega is connected via USB to the computer as there will only be one option to choose from.


Tools
Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
Board: "Arduino/Genuino Mega or Mega 2560"
Processor: "ATmega2560 (Mega 2560)"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader

4) Implement the code as shown in Appendix N into the IDE and select the ‘upload’ button as shown below



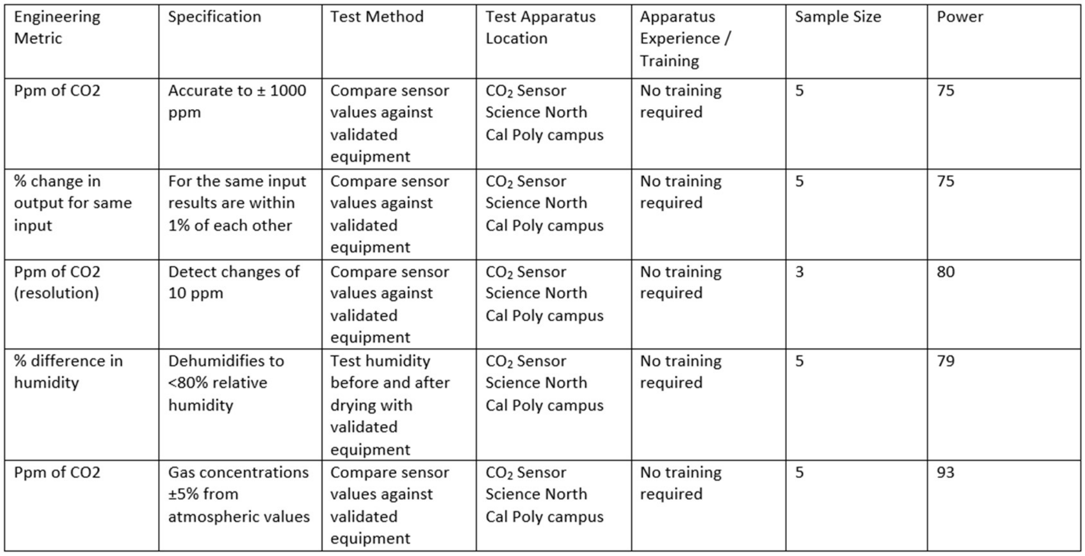
```
////////////////////////////////////////////////////////////////////
// Accelerated Wear Dehumidification Chamber Sensors and Data Logging
// California Polytechnic State University, San Luis Obispo - Biomedical Engineering
// Senior Design - Fall 2018/Winter 2019
// Team: Daniel Alanis, Kian Ashoubi, Tristan Frisella
////////////////////////////////////////////////////////////////////
#include <openGLCD.h>
#include <Wire.h>
#include "SHTSensor.h"
#include "RTClib.h"
#include <SPI.h>
#include <SD.h>
#define TCAADDR 0x70

////////////////////////////////////// Function to use Multiplexer
void tcaselect(uint8_t i) {
  if (i > 7) return;       // Because there are only 7 locations on the multiplexer, calling tcaselect(#) above 7 will cance

  Wire.beginTransmission(TCAADDR);   // Begin wire transmission to multiplexer
  Wire.write(1 << i);                // Shifts bits to switch which location you're connected to on the multiplexer
  Wire.endTransmission();            // End wire transmission to multiplexer
}
////////////////////////////////////// Error function for SD Card
void error(char *str)
  {
  Serial.print("error: ");
  Serial.println(str);
  }
const int chipSelect = 10;    // chipselect for the shield, therefore it's using pin 10
```

6.4 Divergence Between Final Design and Final Functional Prototype

When comparing the Concept Design 1 as shown in section 4.1 to our final functional prototype, we see there are several deviations within the functional prototype. Our conceptual design displays a flow meter attached in series with all the other sensors, but this idea was excluded from the final design as flow will be controlled by an external source. Our functional prototype has a temperature and humidity sensor on both the inlet and outlet, unlike the conceptual design. The purpose of having two temperature and humidity sensors is to verify that our dehumidification chamber is not malfunctioning. The final divergence between the two is that our functional prototype has two CO_2 sensors in series in the outlet unlike the conceptual design. The design choice of incorporating two CO_2 sensors was as a failsafe; if both sensors are reporting drastically different CO_2 readings, this will be an indicator to the user that a sensor is malfunctioning.

7.0 IQ/OQ/PQ

7.1 DOE

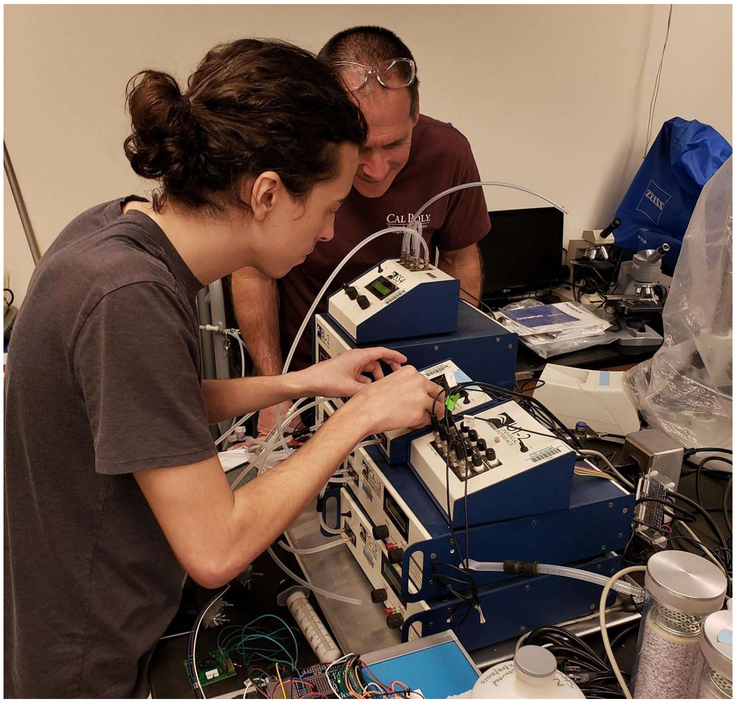


Engineering Metric	Specification	Test Method	Test Apparatus Location	Apparatus Experience / Training	Sample Size	Power
Ppm of CO2	Accurate to ± 1000 ppm	Compare sensor values against validated equipment	CO_2 Sensor Science North Cal Poly campus	No training required	5	75
% change in output for same input	For the same input results are within 1% of each other	Compare sensor values against validated equipment	CO_2 Sensor Science North Cal Poly campus	No training required	5	75
Ppm of CO2 (resolution)	Detect changes of 10 ppm	Compare sensor values against validated equipment	CO_2 Sensor Science North Cal Poly campus	No training required	3	80
% difference in humidity	Dehumidifies to <80% relative humidity	Test humidity before and after drying with validated equipment	CO_2 Sensor Science North Cal Poly campus	No training required	5	79
Ppm of CO2	Gas concentrations ±5% from atmospheric values	Compare sensor values against validated equipment	CO_2 Sensor Science North Cal Poly campus	No training required	5	93

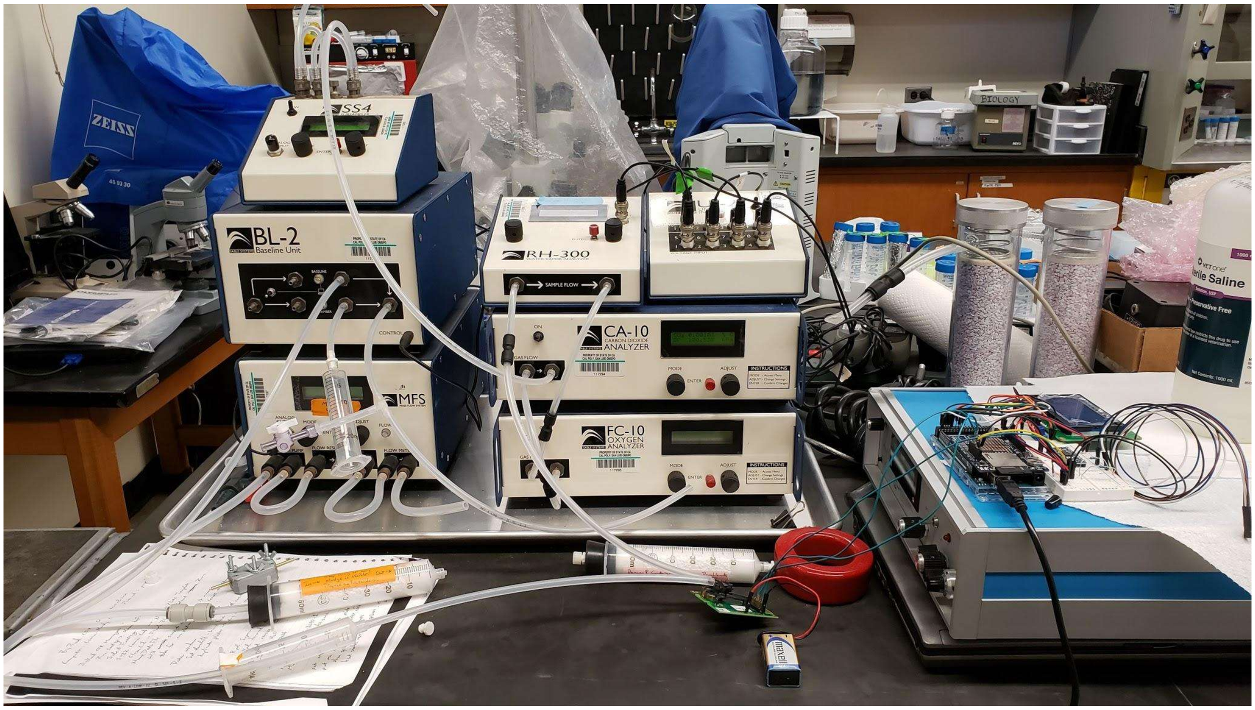
7.2 Verification and Validation

Test plan - CO_2

1. Obtain samples of gas at known concentrations of 0% and 0.5% CO_2 along with samples of unknown concentrations at or below 10% CO_2.
2. Power on the Sable Systems CA-10 carbon dioxide analyzer and allow it to warm up/perform any necessary calibration before testing as shown below.



3. Attach the unvalidated CO_2 meter in line with the validated meter and start recording the data from the unvalidated sensor as shown below.



4. Push the gas samples through both sensors until a steady value is achieved, continue sampling for 30 seconds.

5. Save the recorded data and allow the sensor to return to baseline before pushing the next sample gas and repeat the above steps.

A graphical display of this data can be seen in Figure 10.

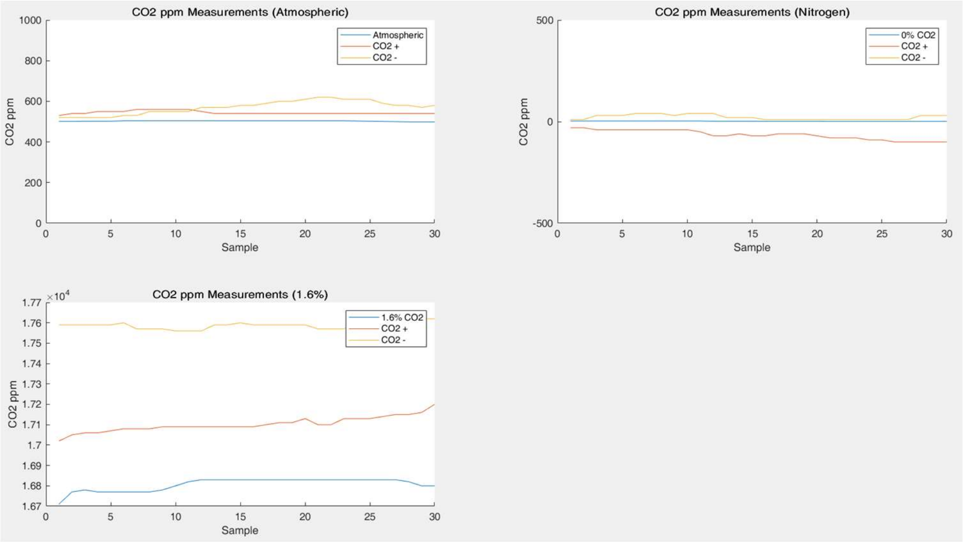
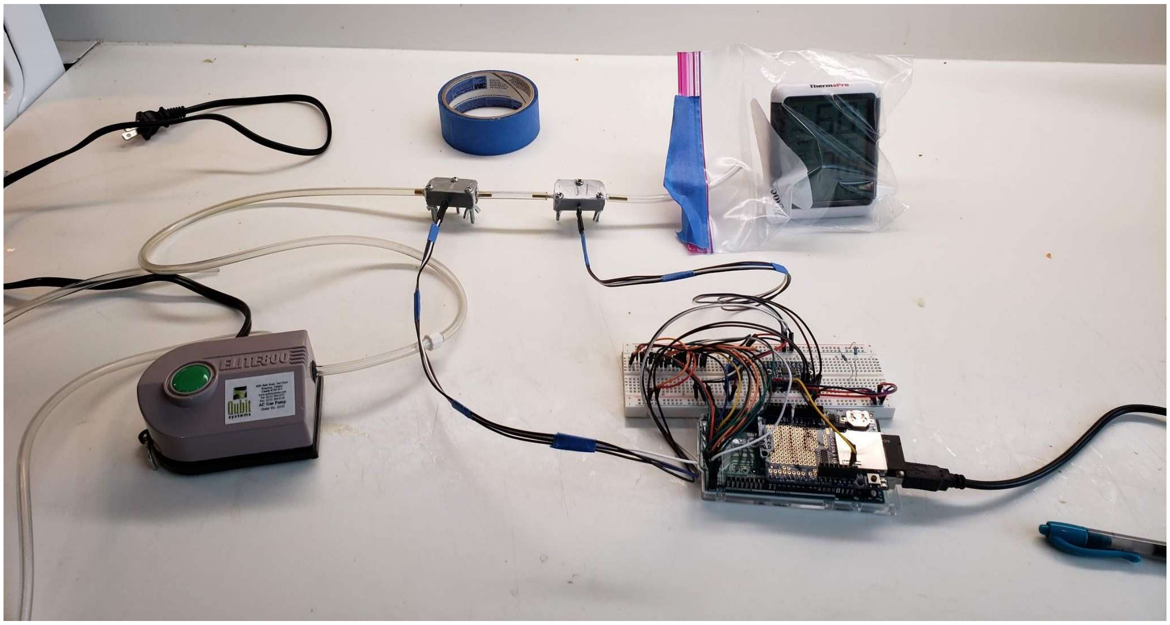


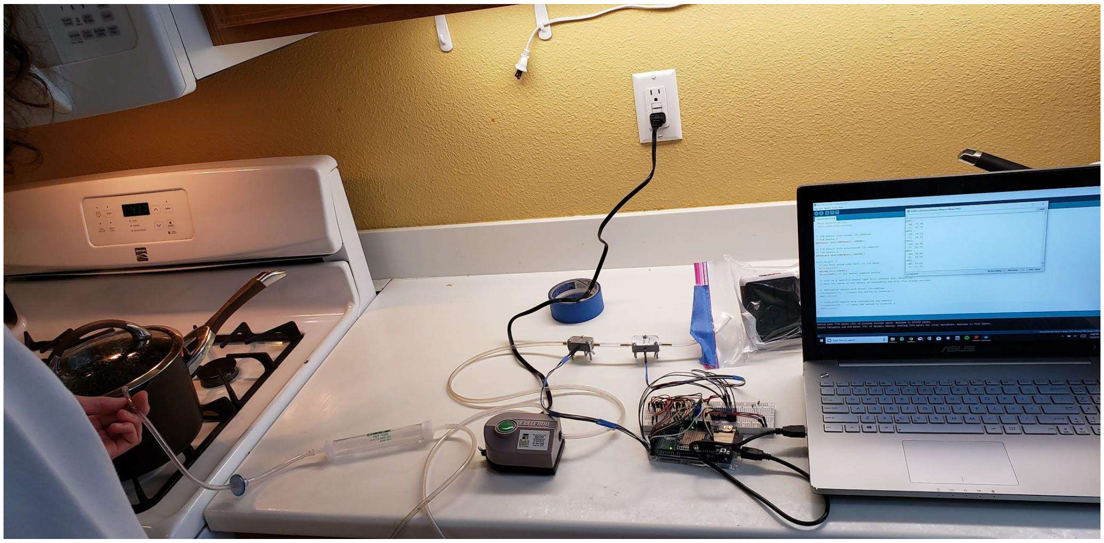
Figure 10: Graph of CO_2 Data, limits have been set to specification range

Test plan - RH/T

1. Generate samples of gasses at varying humidities: ambient, low-humidity (from a dehumidifier), and high-humidity (from a gas bubbler).
2. Power on the ThermoPro handheld digital RH/T sensor and allow it to warm up/perform any necessary calibration before testing.
3. Attach the unvalidated RH/T sensor in line with the validated sensor and begin recording the data from the unvalidated sensor as shown below.



4. Push the gas samples through both sensors until a steady value is achieved, continue sampling for 30 seconds as shown below.



5. Save the recorded data and allow the sensor to return to baseline before pushing the next sample gas and repeat the above steps.

A graphical display of the absolute humidity data can be seen in Figure 11.

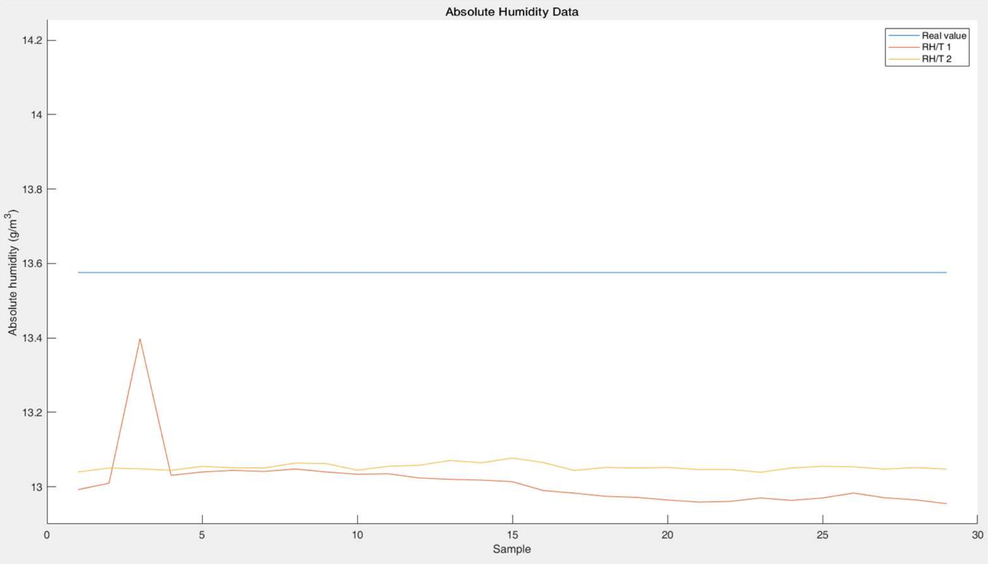


Figure 11: Absolute Humidity Data, limits have been set to specification range

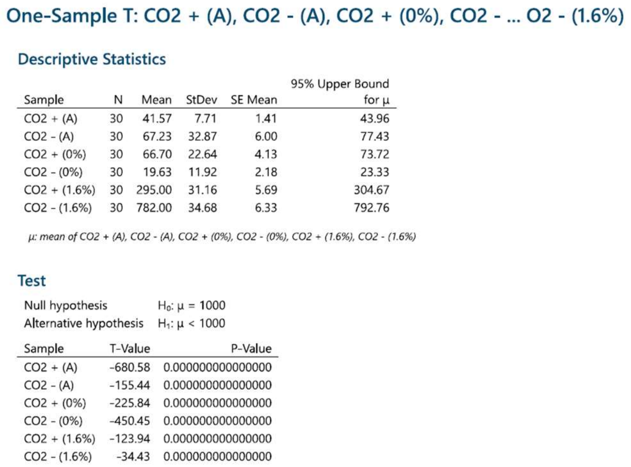
Test plan - Dehumidifier

1. Obtain samples of gasses at ambient and high-humidity
2. Obtain/warm up/calibrate 2 validated relative humidity sensors
3. Attach the sensors to both sides of the dehumidifying chamber
4. Push the gas samples until a steady value is achieved, continue sampling for 10 seconds.
5. Record the difference in humidity between both samples as well as the humidity at the outflow of the dehumidifying chamber.

6. Repeat the above steps for the other gas samples.

Statistical analysis

1. The analysis of the CO_2 data is to be performed by taking the difference of the unvalidated sensor output and the validated sensor and performing a one tailed t-test with an H0: $\mu = 1000$ and Ha: $\mu < 1000$ with an $\alpha=0.05$
2. The analysis of the relative humidity data had to be performed by first converting the RH/T data to absolute humidity using the following equation:$Absolute\ Humidity\ (\frac{g}{m^3}) = \frac{6.112 \cdot e^{\left[\frac{17.67T}{(T+243.5)}\right]} \cdot rh \cdot 2.1674}{(273.15+T)}$
   The difference between the data from the validated and unvalidated sensor was taken. A one tailed t-test with H0: $\mu = 0.678773$ (5% of the validated sensor value) and Ha: $\mu < 0.678773$ with an $\alpha=0.05$
3. Validation of the Dehumidification chamber was performed recording the input and output humidities and confirming that the output was below the threshold value set (80% RH)



One-Sample T: CO2 + (A), CO2 - (A), CO2 + (0%), CO2 - ... O2 - (1.6%)

Descriptive Statistics

Sample	N	Mean	StDev	SE Mean	95% Upper Bound for μ
CO2 + (A)	30	41.57	7.71	1.41	43.96
CO2 - (A)	30	67.23	32.87	6.00	77.43
CO2 + (0%)	30	66.70	22.64	4.13	73.72
CO2 - (0%)	30	19.63	11.92	2.18	23.33
CO2 + (1.6%)	30	295.00	31.16	5.69	304.67
CO2 - (1.6%)	30	782.00	34.68	6.33	792.76

μ: mean of CO2 + (A), CO2 - (A), CO2 + (0%), CO2 - (0%), CO2 + (1.6%), CO2 - (1.6%)

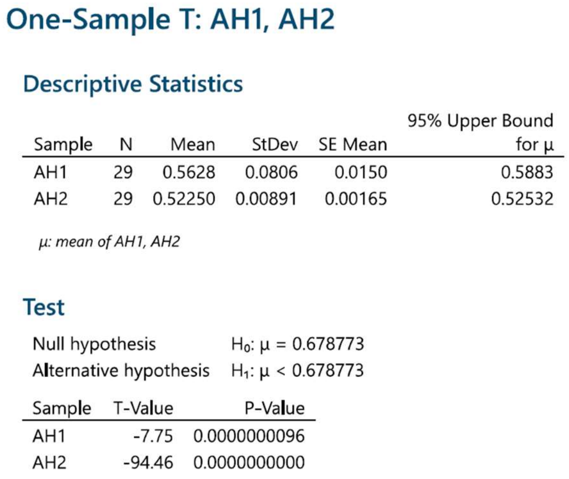
Test

Null hypothesis H_0: μ = 1000
Alternative hypothesis H_1: μ < 1000

Sample	T-Value	P-Value
CO2 + (A)	-680.58	0.000000000000000
CO2 - (A)	-155.44	0.000000000000000
CO2 + (0%)	-225.84	0.000000000000000
CO2 - (0%)	-450.45	0.000000000000000
CO2 + (1.6%)	-123.94	0.000000000000000
CO2 - (1.6%)	-34.43	0.000000000000000

This one sample t-test indicates that with the 95% confidence it can be said that the difference between the sensor output and the real CO_2 concentration of a sample gas is less than the set specification of 1000 ppm over the tested range.

The CO_2 sensor resolution is set by the manufacturer, the resolution of 10 ppm of the sensor exceeds the set specification of 100 ppm.



```
One-Sample T: AH1, AH2

Descriptive Statistics

                                               95% Upper Bound
Sample   N     Mean     StDev   SE Mean                  for μ
AH1     29   0.5628    0.0806    0.0150                 0.5883
AH2     29  0.52250   0.00891   0.00165                0.52532

μ: mean of AH1, AH2

Test

Null hypothesis          H₀: μ = 0.678773
Alternative hypothesis   H₁: μ < 0.678773

Sample   T-Value       P-Value
AH1        -7.75  0.0000000096
AH2       -94.46  0.0000000000
```

The relative humidity sensors were validated by converting their relative humidity output to absolute humidity (since relative humidity is a function of temperature, and temperature changed significantly between sensors) and taking the difference between measured and known humidity values and showing this difference was within 5% of the known humidity value.

No statistical analysis was performed on the dehumidification chamber, this specification was validated by pushing samples of varying humidity and confirming that the output sample was below 80% relative humidity.

The specifications for flow rate and chamber volume were based off of product specifications. Components had ideal flow rates approximately at 0.2 L/min and volume was minimized by connecting components through tubing.

8.0 Conclusions and Recommendations

8.1 Recommendations

For the next iteration of this device, implementation of a heat sink may be incorporated to further reduce the risk of components of the device from overheating. This is not expected to be an issue with this design but will be added as an additional safety feature. Additional sensors may be utilized such as a flow rate transducer since it was observed during testing that flow rates can affect the output of the sensors.

8.2 Conclusions

Through statistical analysis as shown in section 7.2, all of our sensors passed their threshold criteria of accuracy as compared to validated sensors. Therefore, our dehumidification chamber has been validated to be an effective method of measuring CO_2 generation of degraded plastic in an accelerated aged environment as well as being an effective method for measuring and removing humidity from gas samples. This apparatus is now qualified to be implemented in ASTM D5338, D5988, and ISO 14855. Although more work is required to create an accelerated aged environment, our equipment is sufficient enough to take gas samples from an outside source and measure humidity, temperature, and CO_2 in parts per million.

9.0 Acknowledgments

Thank you to the Biomedical Engineering, Industrial Technology and Packaging, and Biological Science Departments here at California Polytechnic State University, San Luis Obispo for giving us the opportunity to learn industry skill sets in a classroom setting. A special thanks to Rob Brewster for his outstanding mentorship throughout the past six months, Dr. Jason Blank for providing us testing equipment to further our senior design, and to Drs. Michael Whitt and Christopher Heylman for simulating a well thought out experience in bringing a medical device to market.

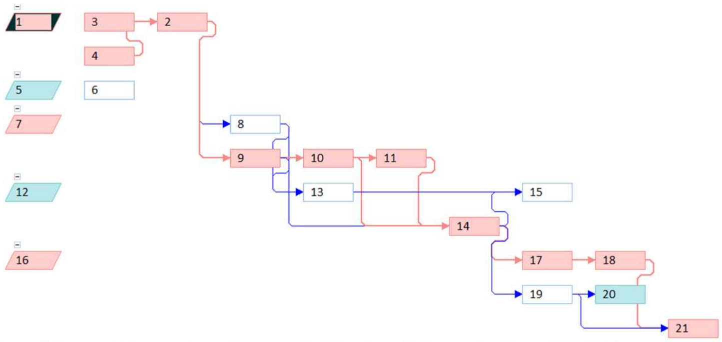
10.0 Appendices

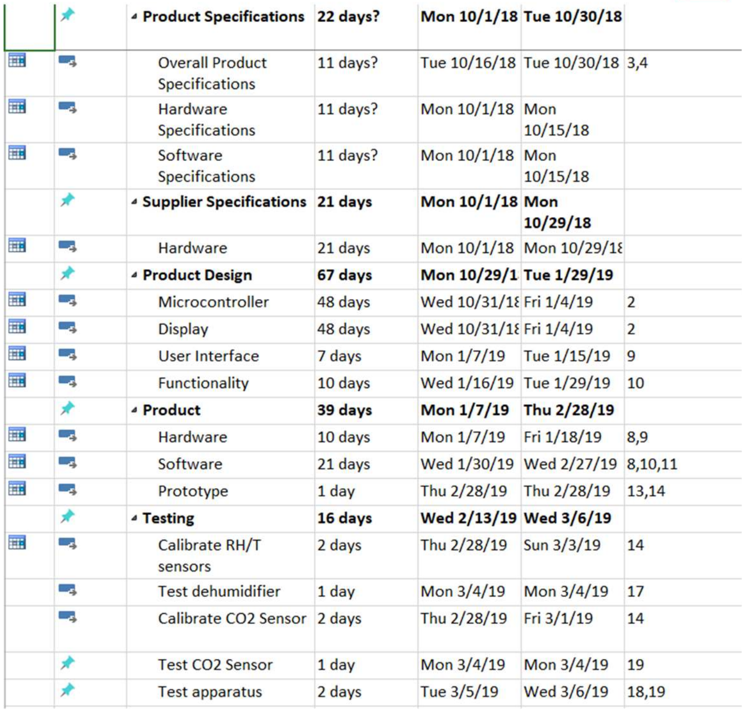
10.1 Appendix A: References

References

1. "Global Greenhouse Gas Emissions Data." EPA, Environmental Protection Agency, 13 Apr. 2017, www.epa.gov/ghgemissions/global-greenhouse-gas-emissions-data.
2. Arnaud, Rene. *Device for Accelerated Photo-Aging of Materials Containing Polymers*. 17 Oct. 1989.
3. Cuddihy, Edward F. *Predictive Aging of Polymers*. 18 Sept. 1990.
4. Yap, Darren Y. K. *Accelerated Life Testing Device and Method*. 23 Feb. 2016.
5. 黄美蓉 . *Aging-Resistant PA (Polyamide) -PPS (Polyphenylene Sulfide) Plastic Alloy*. 20 Jan. 2016.
6. Delabroye, Christine. *Dimensionally-Stable Propylene Polymer Foam with Improved Thermal Aging*. 6 Jan. 2005.
7. Thitislip, Lijchavengkul *Development of an automatic laboratory-scale respirometric system to measure polymer biodegradability.* 24 May 2016.
8. Hansol, Kim *Effect of test parameters on degradation on polyurethane elastomer for accelerated life testing.* 15 August 2014.
9. Lu, Liu *Degradation process investigation of thermoplastic polyurethane elastomer in supercritical methanol.* 13 September 2013.
10. Vesa, Saikko *High frequency circular translation pin-on-disk method for accelerated wear testing of ultrahigh molecular weight polyethylene as a bearing material in total hip arthroplasty.* 24 November 2014.
11. Webb, Hayden *Plastic Degradation and Its Environmental Implications with Special reference to Poly(ethylene terephthalate).* 28 December 2012.
12. "All About Nafion." Perma Pure LLC, www.permapure.com/resources/all-about-nafion-and-faq/.
13. "Types of Desiccants and Desiccant Media - Zeolite, Silica Gel, Activated Alumina - Drytech." Drytech Inc., www.drytechinc.com/types-of-desiccant/.
14. "Number of U.S. Colleges and Universities and Degrees Awarded, 2005" Infoplease, https://www.infoplease.com/us/higher-education/number-us-colleges-and-universities-and-degrees-awarded-2005.
15. "Frequently Asked Questions" ASTM International, https://www.astm.org/cms/drupal-7.51/content/frequently-asked-questions

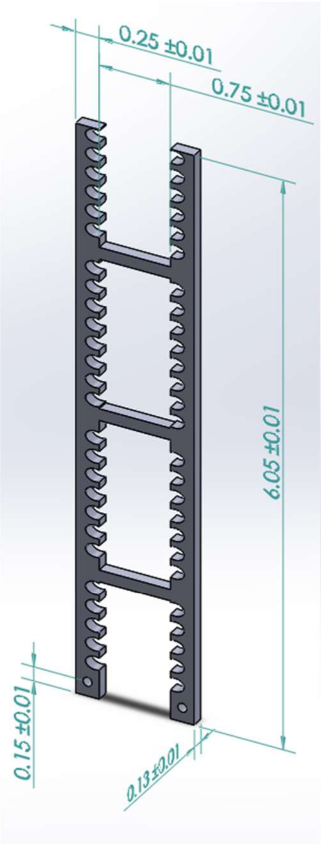
10.2 Appendix B: Project Plan (PERT Chart)

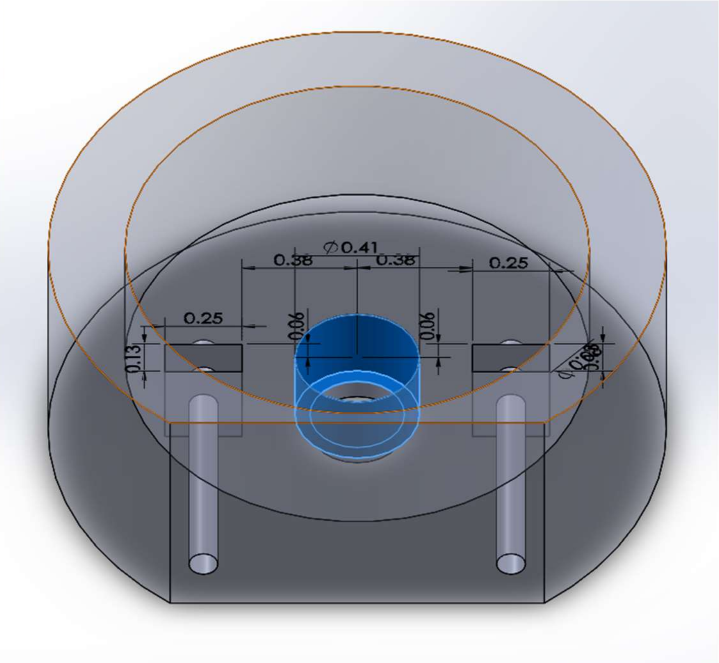


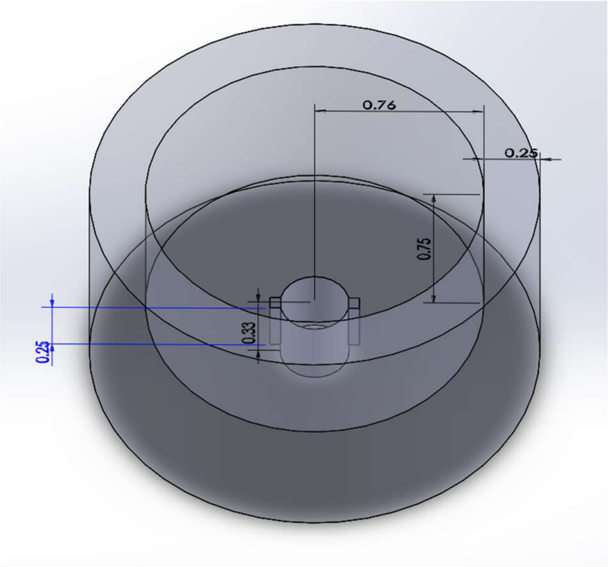


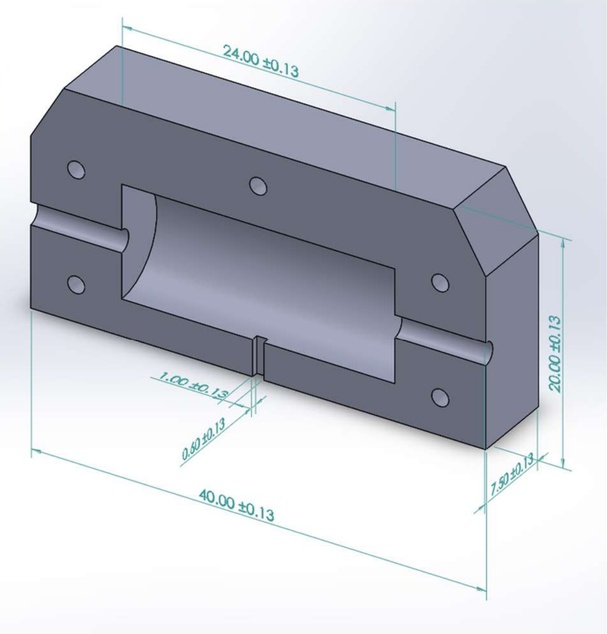
Product Specifications	**22 days?**	**Mon 10/1/18**	**Tue 10/30/18**	
Overall Product Specifications	11 days?	Tue 10/16/18	Tue 10/30/18	3,4
Hardware Specifications	11 days?	Mon 10/1/18	Mon 10/15/18	
Software Specifications	11 days?	Mon 10/1/18	Mon 10/15/18	
Supplier Specifications	**21 days**	**Mon 10/1/18**	**Mon 10/29/18**	
Hardware	21 days	Mon 10/1/18	Mon 10/29/18	
Product Design	**67 days**	**Mon 10/29/1**	**Tue 1/29/19**	
Microcontroller	48 days	Wed 10/31/18	Fri 1/4/19	2
Display	48 days	Wed 10/31/18	Fri 1/4/19	2
User Interface	7 days	Mon 1/7/19	Tue 1/15/19	9
Functionality	10 days	Wed 1/16/19	Tue 1/29/19	10
Product	**39 days**	**Mon 1/7/19**	**Thu 2/28/19**	
Hardware	10 days	Mon 1/7/19	Fri 1/18/19	8,9
Software	21 days	Wed 1/30/19	Wed 2/27/19	8,10,11
Prototype	1 day	Thu 2/28/19	Thu 2/28/19	13,14
Testing	**16 days**	**Wed 2/13/19**	**Wed 3/6/19**	
Calibrate RH/T sensors	2 days	Thu 2/28/19	Sun 3/3/19	14
Test dehumidifier	1 day	Mon 3/4/19	Mon 3/4/19	17
Calibrate CO2 Sensor	2 days	Thu 2/28/19	Fri 3/1/19	14
Test CO2 Sensor	1 day	Mon 3/4/19	Mon 3/4/19	19
Test apparatus	2 days	Tue 3/5/19	Wed 3/6/19	18,19

10.3 Cad Drawings

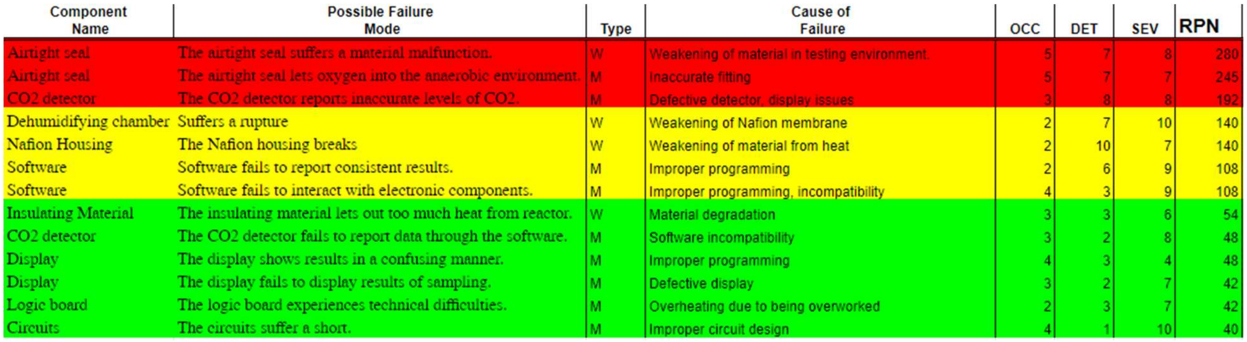




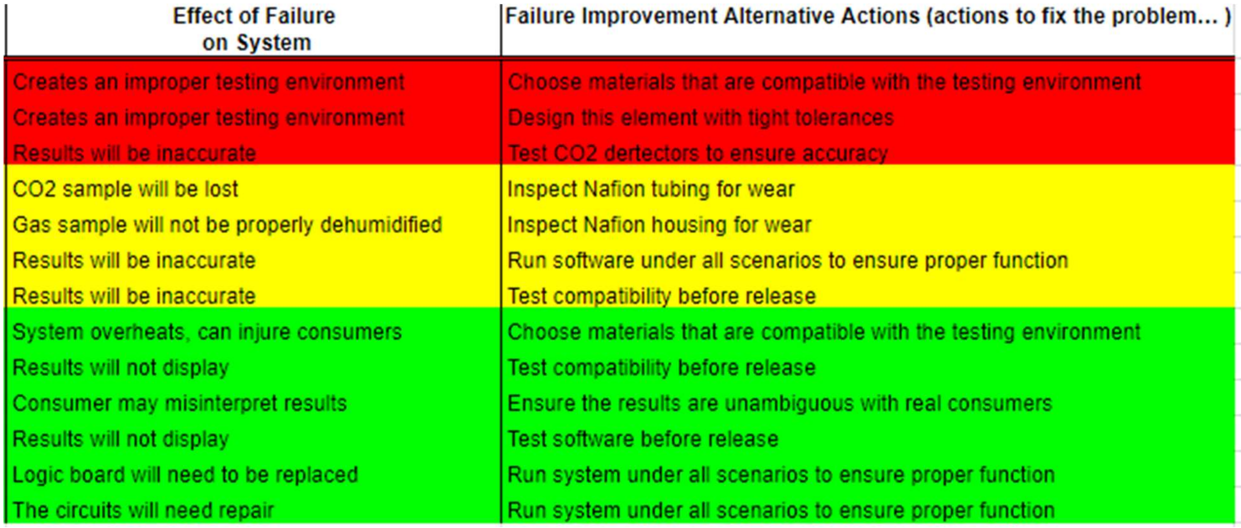




10.4 Appendix D: FMEA, Hazard & Risk Assessment



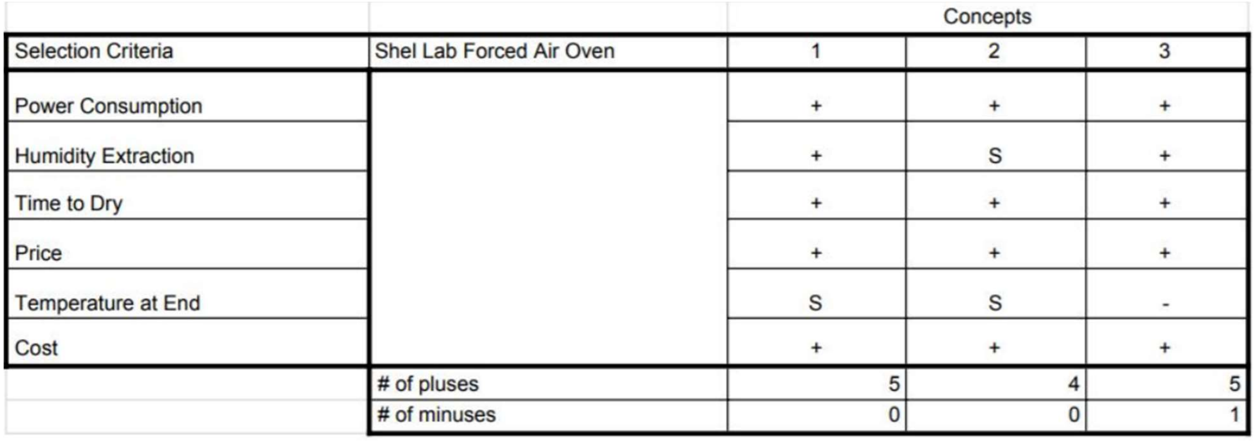
Component Name	Possible Failure Mode	Type	Cause of Failure	OCC	DET	SEV	RPN
Airtight seal	The airtight seal suffers a material malfunction.	W	Weakening of material in testing environment.	5	7	8	280
Airtight seal	The airtight seal lets oxygen into the anaerobic environment.	M	Inaccurate fitting	5	7	7	245
CO2 detector	The CO2 detector reports inaccurate levels of CO2.	M	Defective detector, display issues	3	8	8	192
Dehumidifying chamber	Suffers a rupture	W	Weakening of Nafion membrane	2	7	10	140
Nafion Housing	The Nafion housing breaks	W	Weakening of material from heat	2	10	7	140
Software	Software fails to report consistent results.	M	Improper programming	2	6	9	108
Software	Software fails to interact with electronic components.	M	Improper programming, incompatibility	4	3	9	108
Insulating Material	The insulating material lets out too much heat from reactor.	W	Material degradation	3	3	6	54
CO2 detector	The CO2 detector fails to report data through the software.	M	Software incompatibility	3	2	8	48
Display	The display shows results in a confusing manner.	M	Improper programming	4	3	4	48
Display	The display fails to display results of sampling.	M	Defective display	3	2	7	42
Logic board	The logic board experiences technical difficulties.	M	Overheating due to being overworked	2	3	7	42
Circuits	The circuits suffer a short.	M	Improper circuit design	4	1	10	40



Effect of Failure on System	Failure Improvement Alternative Actions (actions to fix the problem...)
Creates an improper testing environment	Choose materials that are compatible with the testing environment
Creates an improper testing environment	Design this element with tight tolerances
Results will be inaccurate	Test CO2 dertectors to ensure accuracy
CO2 sample will be lost	Inspect Nafion tubing for wear
Gas sample will not be properly dehumidified	Inspect Nafion housing for wear
Results will be inaccurate	Run software under all scenarios to ensure proper function
Results will be inaccurate	Test compatibility before release
System overheats, can injure consumers	Choose materials that are compatible with the testing environment
Results will not display	Test compatibility before release
Consumer may misinterpret results	Ensure the results are unambiguous with real consumers
Results will not display	Test software before release
Logic board will need to be replaced	Run system under all scenarios to ensure proper function
The circuits will need repair	Run system under all scenarios to ensure proper function

Many of the above failure modes were examined with the entire assembly (including the bioreactor) in mind. For the scope of the project described in this report many of the above failure modes are outside the scope of this design. The most significant failure mode that falls under the scope of this design would be the accuracy of the CO_2 sensor. This risk was mitigated by having a clearly defined specification for a functioning CO_2 sensor. To mitigate risks further, the CO_2 sensors were tested against each other using various measured CO_2 samples to ensure sensor accuracy. Failure warnings regarding the Nafion tubing were mitigated by implementing color-indicating silica gel desiccant into the design. The desiccant ensures further dehumidifying of air samples as they pass through the Nafion tubing chamber. Desiccant is easily removable once color-changing indicates that it requires drying.

10.5 Appendix E: Pugh Chart



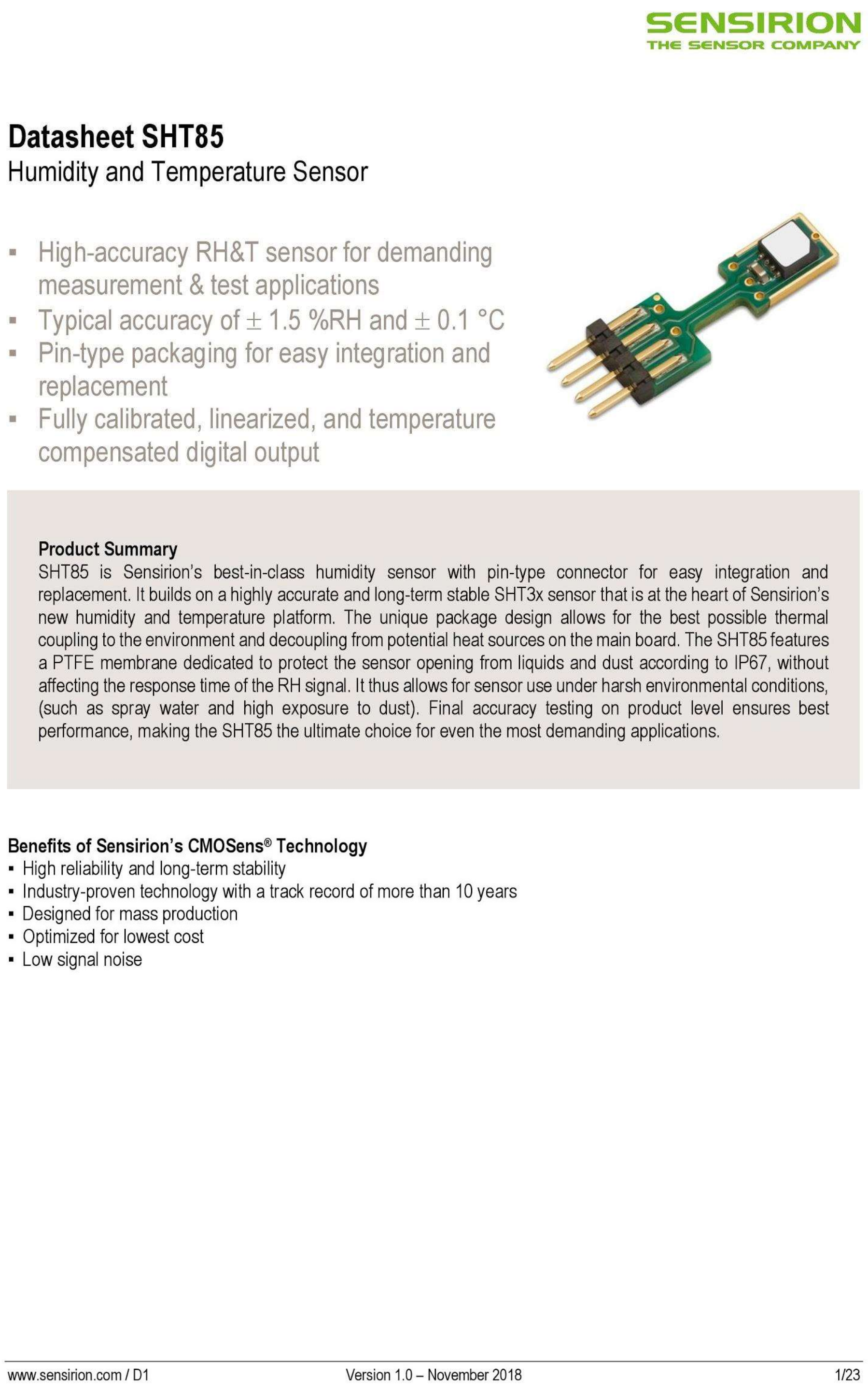
		Concepts		
Selection Criteria	Shel Lab Forced Air Oven	1	2	3
Power Consumption		+	+	+
Humidity Extraction		+	S	+
Time to Dry		+	+	+
Price		+	+	+
Temperature at End		S	S	-
Cost		+	+	+
	# of pluses	5	4	5
	# of minuses	0	0	1

10.6 Appendix F: Vendor Information

CO2 Meter:
CO2Meter.com
131 Business Center Drive
Ormond Beach, FL
32174 USA
(877) 678-4259

Adafruit:
Adafruit.com

10.7 Appendix G: Sensirion Digital Humidity Sensor SHT85 (RH/T) Datasheet



SENSIRION
THE SENSOR COMPANY

Datasheet SHT85

Humidity and Temperature Sensor

- High-accuracy RH&T sensor for demanding measurement & test applications
- Typical accuracy of ± 1.5 %RH and ± 0.1 °C
- Pin-type packaging for easy integration and replacement
- Fully calibrated, linearized, and temperature compensated digital output

Product Summary
SHT85 is Sensirion's best-in-class humidity sensor with pin-type connector for easy integration and replacement. It builds on a highly accurate and long-term stable SHT3x sensor that is at the heart of Sensirion's new humidity and temperature platform. The unique package design allows for the best possible thermal coupling to the environment and decoupling from potential heat sources on the main board. The SHT85 features a PTFE membrane dedicated to protect the sensor opening from liquids and dust according to IP67, without affecting the response time of the RH signal. It thus allows for sensor use under harsh environmental conditions, (such as spray water and high exposure to dust). Final accuracy testing on product level ensures best performance, making the SHT85 the ultimate choice for even the most demanding applications.

Benefits of Sensirion's CMOSens® Technology

- High reliability and long-term stability
- Industry-proven technology with a track record of more than 10 years
- Designed for mass production
- Optimized for lowest cost
- Low signal noise

www.sensirion.com / D1 Version 1.0 – November 2018 1/23

Content

1 Humidity and Temperature Sensor Specifications

Relative Humidity

Parameter	Conditions	Value	Units
Accuracy tolerance[1]	Typ.	±1.5	%RH
	Max.	see Figure 1	-
Repeatability[2]	Low, typ.	0.21	%RH
	Medium, typ.	0.15	%RH
	High, typ.	0.08	%RH
Resolution	Typ.	0.01	%RH
Hysteresis	At 25°C	±0.8	%RH
Specified range[3]	Non-condensing environment[4]	0 to 100	%RH
Response time[5]	τ 63%	8[6]	s
Long-term drift[7]	Typ.	<0.25	%RH/y

Table 1: Humidity sensor specifications

Temperature

Parameter	Conditions	Value	Units
Accuracy tolerance[1]	Typ., 20°C to 50 °C	±0.1	°C
	Max.	see Figure 2	-
Repeatability[2]	Low, typ.	0.15	°C
	Medium, typ.	0.08	°C
	High, typ.	0.04	°C
Resolution	Typ.	0.01	°C
Operating range	-	–40 to 105[8]	°C
Response time[9]	τ 63%	>2	s
Long-term drift	Typ.	<0.03	°C/y

Table 2: Temperature sensor specifications

[1] For definition of typ. and max. accuracy tolerance, please refer to the document “Sensirion Humidity Sensor Specification Statement”.

[2] The stated repeatability is 3 times the standard deviation (3σ) of multiple consecutive measurement values at constant conditions and is a measure for the noise on the physical sensor output.

[3] Specified range refers to the range for which the humidity sensor specification is guaranteed.

[4] Condensation shall be avoided because of risk of corrosion and leak currents on the PCB. For details about recommended humidity and temperature operating range, please refer to Section 1.2.

[5] Time for achieving 63% of a humidity step function, valid at 25°C and 1 m/s airflow. Humidity response time in the application depends on the design-in of the sensor.

[6] With activated ART function (see Section 4.8) the response time can be improved by a factor of 2.

[7] Typical value for operation in normal RH/T operating range. Max. value is < 0.5 %RH/y. Value may be higher in environments with vaporized solvents, out-gassing tapes, adhesives, packaging materials, etc. For more details please refer to Handling Instructions.

[8] All parts, incl. PCB are rated up to 125°C, except for the connector, which is rated for 105°C.

[9] Temperature response time depends on heat conductivity of sensor substrate and design-in of sensor in application.

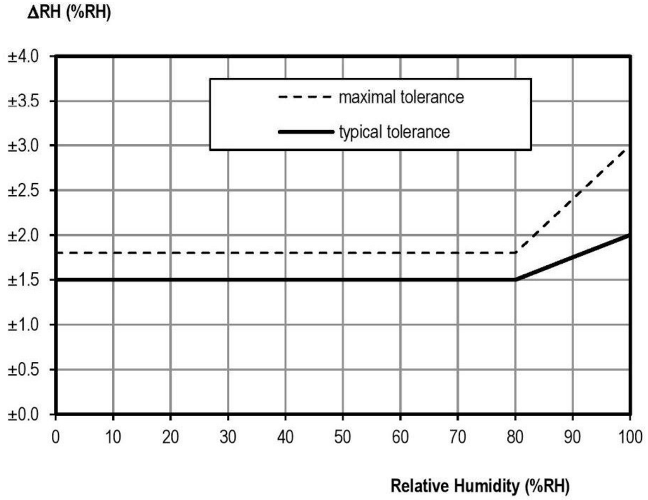


Figure 1: Typical and maximal tolerance for relative humidity in %RH at 25 °C.

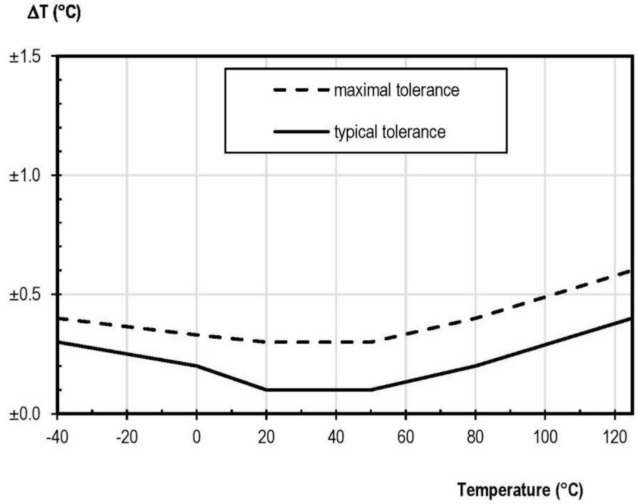


Figure 2: Typical and maximal tolerance for temperature sensor in °C

1.1 RH Accuracy at Various Temperatures

Typical RH accuracy at 25°C is defined in Figure 2. For other temperatures, typical accuracy has been evaluated to be as displayed in Figure 4.

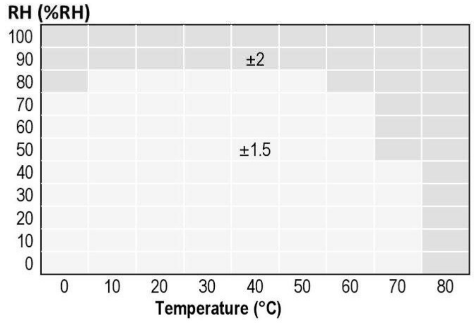


Figure 3: Typical accuracy of relative humidity measurements given in %RH for temperatures 0 – 80°C.

1.2 Recommended Operating Conditions

The sensor shows best performance when operated within recommended normal temperature and humidity range of 5 – 60 °C and 20 – 80 %RH, respectively. Long term exposure to conditions outside normal range, especially at high humidity, may temporarily offset the RH signal (e.g. +3%RH after 60h at >80%RH). After returning into the normal temperature and humidity range, the sensor will slowly come back to calibration state by itself. Prolonged exposure to extreme conditions may accelerate ageing.

To ensure stable operation of the humidity sensor, the conditions described in the document "SHTxx Assembly of SMD Packages", Section "Storage and Handling Instructions" regarding exposure to volatile organic compounds have to be met. Please note as well that this does apply not only to transportation and manufacturing, but also to operation of the SHT85.

2 Electrical Specifications

2.1 Electrical Characteristics

Parameter	Symbol	Conditions	Min	Typ.	Max	Units	Comments
Supply voltage	V_{DD}		2.15	3.3	5.5	V	-
Power-up/down level	V_{POR}		1.8	2.1	2.15	V	-
Slew rate change of the supply voltage	$V_{DD,slew}$		-	-	20	V/ms	Voltage changes on the VDD line between $V_{DD,min}$ and $V_{DD,max}$ should be slower than the maximum slew rate; faster slew rates may lead to reset;
Supply current	I_{DD}	Idle state (single shot mode) T= 25°C	-	0.2	12.0	µA	Current when sensor is not performing a measurement during single shot mode
		Idle state (single shot mode) T= 125°C	-	-	6.0		
		Idle state (periodic data acquisition mode)	-	45	-	µA	Current when sensor is not performing a measurement during periodic data acquisition mode
		Measurement	-	600	1500	µA	Current consumption while sensor is measuring
		Average	-	1.7	-	µA	Average current consumption (operation with one measurement per second at lowest repeatability, single shot mode)
Heater Power	P_{Heater}	Heater running	3.6	-	33	mW	Depending on the supply voltage

Table 3: Electrical specifications, typical values are valid for T=25°C, min. & max. values for T=-40°C … 125°C.

2.2 Timing Specifications

Parameter	Symbol	Conditions	Min.	Typ.	Max.	Units	Comments
Power-up time	t_{PU}	After hard reset, $V_{DD} \geq V_{POR}$	-	0.5	1.5	ms	Time between V_{DD} reaching V_{PU} and sensor entering idle state
Soft reset time	t_{SR}	After soft reset.	-	0.5	1.5	ms	Time between ACK of soft reset command and sensor entering idle state
Measurement duration	$t_{MEAS,l}$	Low repeatability	-	2.5	4.5	ms	The three repeatability modes differ with respect to measurement duration, noise level and energy consumption.
	$t_{MEAS,m}$	Medium repeatability	-	4.5	6.5	ms	
	$t_{MEAS,h}$	High repeatability	-	12.5	15.5	ms	

Table 4: System timing specifications, valid from -40 °C to 125 °C and VDDmin to VDDmax.

2.3 Absolute Minimum and Maximum Ratings

Stress levels beyond those listed in Table 5 may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these conditions cannot be guaranteed. Exposure to the absolute maximum rating conditions for extended periods may affect the reliability of the device. Ratings are only tested each at a time.

Parameter	Rating
Supply voltage, V_{DD}	-0.3 to 6 V
Max voltage on pins (pin 1 (SCL); pin 4 (SDA);	-0.3 to VDD+0. V
Input current on any pin	±100 mA
Operating temperature range	-40 to 105 °C
Storage temperature range[10]	-40 to 105 °C
ESD HBM (human body model)[11]	4 kV
ESD CDM (charge device model)[12]	750 V

Table 5: Absolute maximum ratings.

[10] The recommended storage temperature range is 10-50°C. Please consult the document "SHTxx Handling Instructions" for more information.

[11] According to ANSI/ESDA/JEDEC JS-001-2014; AEC-Q100-002.

[12] According to ANSI/ESD S5.3.1-2009; AEC-Q100-011.

3 Pin Assignment

The SHT85 comes with a 4-pin-type connector, see Table 6.

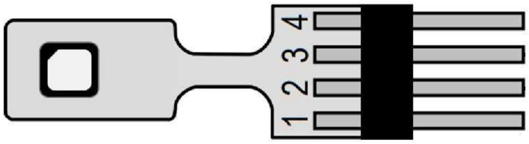
Pin	Name	Comments
1	SCL	Serial clock; input only
2	VDD	Supply voltage; input
3	VSS	Ground
4	SDA	Serial data; input / output
		

Table 6: SHT85 pin assignment (transparent top view). The die pad is internally connected to VSS.

3.1 Power Pins (VDD, VSS)

The electrical specifications of the SHT85 are shown in Table 3. Decoupling of VDD and VSS by a 100nF capacitor is integrated on the front side of the sensor packaging. See Figure 4 for a typical application circuit.

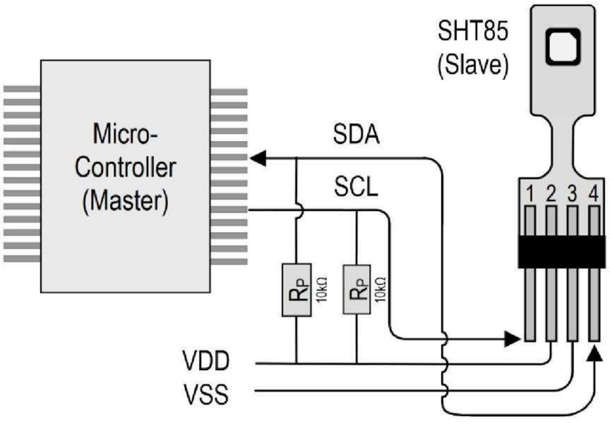


Figure 4: Typical application circuit

3.2 Serial Clock and Serial Data (SCL, SDA)

SCL is used to synchronize the communication between microcontroller and the sensor. The clock frequency can be freely chosen between 0 to 1000 kHz.

The SDA pin is used to transfer data to and from the sensor. Communication with frequencies up to 400 kHz must meet the I2C *Fast Mode*[13] standard. Communication frequencies up to 1 Mhz are supported following the specifications given in Table 20.

[13] http://www.nxp.com/documents/user_manual/UM10204.pdf

4 Operation and Communication

The SHT85 supports I2C fast mode (and frequencies up to 1000 kHz). For detailed information on the I2C protocol, refer to NXP I2C-bus specification[14].

After sending a command to the sensor a minimal waiting time of 1ms is needed before another command can be received by the sensor.

Furthermore, to keep self-heating below 0.1°C, SHT85 should not be active for more than 10% of the time.

All SHT85 commands and data are mapped to a 16-bit address space. Additionally, data and commands are protected with a CRC checksum. This increases communication reliability. The 16 bits commands to the sensor already include a 3 bit CRC checksum. Data sent from and received by the sensor is always succeeded by an 8 bit CRC.

In write direction it is mandatory to transmit the checksum, since the SHT85 only accepts data if it is followed by the correct checksum. In read direction it is left to the master to read and process the checksum.

4.1 I2C Address

The I2C device address is given in Table 7: SHTC85 I^2C device address.

SHT85	Hex. Code	Bin. Code
I^2C address	0x44	100’0100

Table 7: SHTC85 I^2C device address.

4.2 Power-Up and Communication Start

The sensor starts powering-up after reaching the power-up threshold voltage V_{POR} specified in Table 3. After reaching this threshold voltage the sensor needs the time t_{PU} to enter idle state. Once the idle state is entered it is ready to receive commands from the master (microcontroller).

Each transmission sequence begins with a START condition (S) and ends with a STOP condition (P) as described in the I2C-bus specification. Whenever the sensor is powered up, but not performing a measurement or communicating, it automatically enters idle state for energy saving. This idle state cannot be controlled by the user.

4.3 Starting a Measurement

A measurement communication sequence consists of a START condition, the I2C write header (7-bit I2C device address plus 0 as the write bit) and a 16-bit measurement command. The proper reception of each byte is indicated by the sensor. It pulls the SDA pin low (ACK bit) after the falling edge of the 8th SCL clock to indicate the reception. A complete measurement cycle is depicted in Table 8.

With the acknowledgement of the measurement command, the SHT85 starts measuring humidity and temperature.

4.4 Measurement Commands for Single Shot Data Acquisition Mode

In this mode one issued measurement command triggers the acquisition of *one data pair*. Each data pair consists of one 16-bit temperature and one 16-bit humidity value (in this order). During transmission each data value is always followed by a CRC checksum, see Section 4.5.

In single shot mode different measurement commands can be selected. The 16-bit commands are shown in Table 8. They differ with respect to repeatability (low, medium and high).

[14] http://www.nxp.com/documents/user_manual/UM10204.pdf

The repeatability setting influences the measurement duration and thus the overall energy consumption of the sensor. This is explained in Section 2.

Condition	Hex. code	
Repeatability	MSB	LSB
High	0x24	00
Medium		0B
Low		16
e.g. 0x2400: high repeatability measurement.		

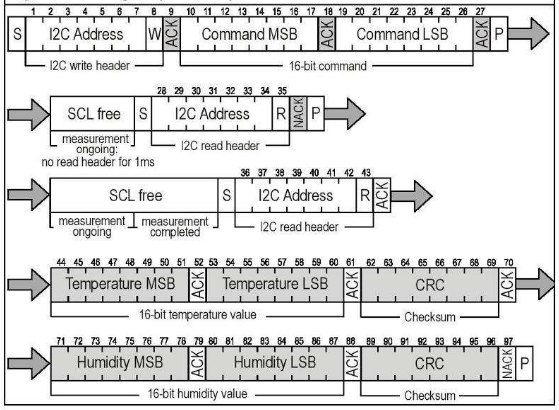


Table 8: Measurement commands in single shot mode. The first "SCL free" block indicates a minimal waiting time of 1ms. (Clear blocks are controlled by the microcontroller, grey blocks by the sensor).

4.5 Readout of Measurement Results for Single Shot Mode

After the sensor has completed the measurement, the master can read the measurement results (pair of RH & T) by sending a START condition followed by an I2C read header.

The sensor responds to a read header with a not acknowledge (NACK), if the measurement is still ongoing and thus no data is present.

If the measurement is completed, the sensor will acknowledge the reception of the read header and send two bytes of data (temperature) followed by one byte CRC checksum and another two bytes of data (relative humidity) followed by one byte CRC checksum. Each byte must be acknowledged by the microcontroller with an ACK condition for the sensor to continue sending data. If the sensor does not receive an ACK from the master after any byte of data, it will not continue sending data.

The sensor will send the temperature value first and then the relative humidity value. After having received the checksum for the humidity value a NACK and stop condition should be sent (see Table 8).

The I2C master can abort the read transfer with a NACK condition after any data byte if it is not interested in subsequent data, e.g. the CRC byte or the second measurement result, in order to save time.

In case the user needs humidity and temperature data but does not want to process CRC data, it is recommended to read the two temperature bytes of data with the CRC byte (without processing the CRC data); after having read the two humidity bytes, the read transfer can be aborted with a with a NACK.

4.6 Measurement Commands for Periodic Data Acquisition Mode

In this mode one issued measurement command yields *a stream of data pairs*. Each data pair consists of one 16-bit temperature and one 16-bit humidity value (in this order).

In periodic mode different measurement commands can be selected. The corresponding 16-bit commands are shown in Table 9. They differ with respect to repeatability (low, medium and high) and data acquisition frequency (0.5, 1, 2, 4 & 10 measurements per second, mps).

The data acquisition frequency and the repeatability setting influences the measurement duration and the current consumption of the sensor. This is explained in Section 2 of this datasheet.

If a measurement command is issued, while the sensor is busy with a measurement (measurement durations see Table 4), it is recommended to issue a break command first (see Section 4.9). Upon reception of the break command the sensor will abort the ongoing measurement and enter the single shot mode.

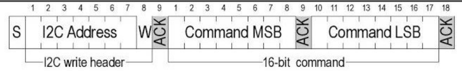
Condition		Hex. code	
Repeatability	mps	MSB	LSB
High	0.5	0x20	32
Medium			24
Low			2F
High	1	0x21	30
Medium			26
Low			2D
High	2	0x22	36
Medium			20
Low			2B
High	4	0x23	34
Medium			22
Low			29
High	10	0x27	37
Medium			21
Low			2A
e.g. 0x2130: 1 high repeatability mps - measurement per second			
			

Table 9: Measurement commands for periodic data acquisition mode (Clear blocks are controlled by the microcontroller, grey blocks by the sensor). N.B.: At the highest mps setting self-heating of the sensor might occur.

4.7 Readout of Measurement Results for Periodic Mode

Transmission of the measurement data can be initiated through the fetch data command shown in Table 10. If no measurement data is present the I2C read header is responded with a NACK (Bit 9 in Table 10) and the communication stops. After the read out command fetch data has been issued, the data memory is cleared, i.e. no measurement data is present.

Command	Hex code
Fetch Data	0x E0 00

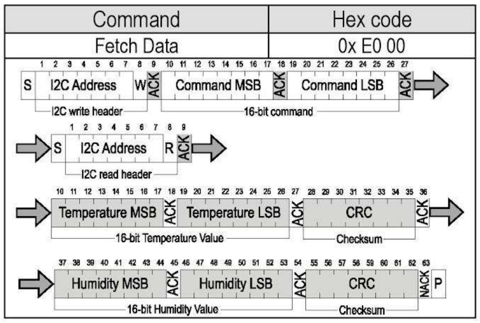


Table 10: Fetch Data command (Clear blocks are controlled by the microcontroller, grey blocks by the sensor).

4.8 ART Command

The ART (accelerated response time) feature can be activated by issuing the command in Table 11. After issuing the ART command the sensor will start acquiring data with a frequency of 4Hz.

The ART command is structurally similar to any other command in Table 9. Hence Section 4.6 applies for starting a measurement, Section 4.7 for reading out data and Section 4.9 for stopping the periodic data acquisition.

Command	Hex Code
Periodic Measurement with ART	0x2B32

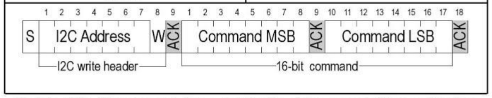


Table 11: Command for a periodic data acquisition with the ART feature (Clear blocks are controlled by the microcontroller, grey blocks by the sensor).

4.9 Break command / Stop Periodic Data Acquisition Mode

The periodic data acquisition mode can be stopped using the break command shown in Table 12. It is recommended to stop the periodic data acquisition prior to sending another command (except Fetch Data command) using the break command. Upon reception of the break command the sensor will abort the ongoing measurement and enter the single shot mode. This takes 1ms.

Command	Hex Code
Break	0x3093

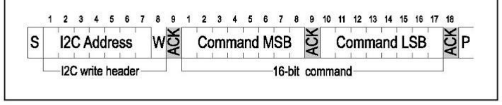


Table 12: Break command (Clear blocks are controlled by the microcontroller, grey blocks by the sensor).

4.10 Reset

A system reset of the SHT85 can be generated externally by issuing a command (soft reset). Additionally, a system reset is generated internally during power-up. During the reset procedure the sensor will not process commands.

Interface Reset

If communication with the device is lost, the following signal sequence will reset the serial interface: While leaving SDA high, toggle SCL nine or more times. This must be followed by a Transmission Start sequence preceding the next command. This sequence resets the interface only. The status register preserves its content.

Soft Reset / Re-Initialization

The SHT85 provides a soft reset mechanism that forces the system into a well-defined state without removing the power supply. When the system is in idle state the soft reset command can be sent to the SHT85. This triggers the sensor to reset its system controller and reloads calibration data from the memory. In order to start the soft reset procedure the command as shown in Table 13 should be sent.

It is worth noting that the sensor reloads calibration data prior to every measurement by default.

Command	Hex Code
Soft Reset	0x30A2

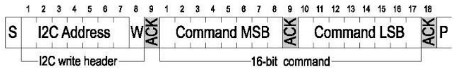


Table 13: Soft reset command (Clear blocks are controlled by the microcontroller, grey blocks by the sensor).

Reset through General Call

Additionally, a reset of the sensor can also be generated using the "general call" mode according to I2C-bus specification[14]. It is important to understand that a reset generated in this way is not device specific. All devices on the same I2C bus that support the general call mode will perform a reset. Additionally, this command only works when the sensor is able to process I2C commands. The appropriate command consists of two bytes and is shown in Table 14.

Command	Code
Address byte	0x00
Second byte	0x06
Reset command using the general call address	0x0006

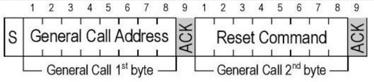


Table 14: Reset through the general call address (Clear blocks are controlled by the microcontroller, grey blocks by the sensor).

Hard Reset

A hard reset is achieved by switching the supply voltage to the VDD Pin off and then on again. In order to prevent powering the sensor over the ESD diodes, the voltage to pins 1 (SCL) and 4 (SDA) also needs to be removed.

4.11 Heater

The SHT85 is equipped with an internal heater, which is meant for plausibility checking only. The temperature increase achieved by the heater depends on various parameters and lies in the range of a few degrees centigrade. It can be switched on and off by command, see table below. The status is listed in the status register. After a reset the heater is disabled (default condition).

Command	Hex Code	
	MSB	LSB
Heater Enable	0x30	6D
Heater Disabled		66

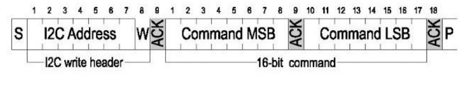


Table 15: Heater command (Clear blocks are controlled by the microcontroller, grey blocks by the sensor).

4.12 Status Register

The status register contains information on the operational status of the heater, the alert mode and on the execution status of the last command and the last write sequence. The command to read out the status register is shown in Table 16 whereas a description of the content can be found in Table 17.

Command	Hex code
Read Out of status register	0xF32D

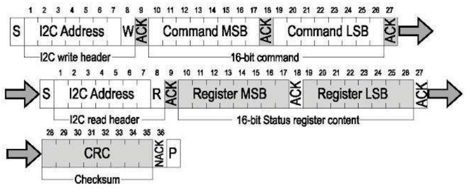


Table 16: Command to read out the status register (Clear blocks are controlled by the microcontroller, grey blocks by the sensor).

Bit	Field description	Default value
15	Alert pending status '0': no pending alerts '1': at least one pending alert	‘1’
14	Reserved	‘0’
13	Heater status ‘0’ : Heater OFF ‘1’ : Heater ON	‘0’
12	Reserved	‘0’
11	RH tracking alert ‘0’ : no alert ‘1’ . alert	‘0
10	T tracking alert ‘0’ : no alert ‘1’ . alert	‘0’
9:5	Reserved	‘xxxxx’
4	System reset detected '0': no reset detected since last ‘clear status register’ command '1': reset detected (hard reset, soft reset command or supply fail)	‘1’
3:2	Reserved	‘00’
1	Command status '0': last command executed successfully '1': last command not processed. It was either invalid, failed the integrated command checksum	‘0’
0	Write data checksum status '0': checksum of last write transfer was correct '1': checksum of last write transfer failed	‘0’

Table 17: Description of the status register.

Clear Status Register

All flags (Bit 15, 11, 10, 4) in the status register can be cleared (set to zero) by sending the command shown in Table 18.

Command	Hex Code
Clear status register	0x 30 41

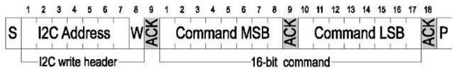


Table 18: Command to clear the status register (Clear blocks are controlled by the microcontroller, grey blocks by the sensor).

4.13 Checksum Calculation

The 8-bit CRC checksum transmitted after each data word is generated by a CRC algorithm. Its properties are displayed in Table 19. The CRC covers the contents of the two previously transmitted data bytes. To calculate the checksum only these two previously transmitted data bytes are used.

Property	Value
Name	CRC-8
Width	8 bit
Protected data	read and/or write data
Polynomial	0x31 ($x^8 + x^5 + x^4 + 1$)
Initialization	0xFF
Reflect input	False
Reflect output	False
Final XOR	0x00
Examples	CRC (0xBEEF) = 0x92

Table 19: I2C CRC properties.

4.14 Conversion of Signal Output

Measurement data is always transferred as 16-bit values (unsigned integer). These values are already linearized and compensated for temperature and supply voltage effects. Converting those raw values into a physical scale can be achieved using the following formulas.

Relative humidity conversion formula (result in %RH):

$$RH = 100 \cdot \frac{S_{RH}}{2^{16} - 1}$$

Temperature conversion formula (result in °C & °F):

$$T\left[{}^\circ C\right] = -45 + 175 \cdot \frac{S_T}{2^{16} - 1}$$

$$T\left[{}^\circ F\right] = -49 + 315 \cdot \frac{S_T}{2^{16} - 1}$$

S_{RH} and S_T denote the raw sensor output for humidity and temperature, respectively. The formulas work only correctly when S_{RH} and S_T are used in decimal representation.

4.15 Communication Timing

Parameter	Symbol	Conditions	Min.	Typ.	Max.	Units	Comments
SCL clock frequency	f_{SCL}		0	-	1000	kHz	
Hold time (repeated) START condition	$t_{HD;STA}$	After this period, the first clock pulse is generated	0.24	-	-	µs	
LOW period of the SCL clock	t_{LOW}		0.53	-	-	µs	
HIGH period of the SCL clock	t_{HIGH}		0.26	-	-	µs	
SDA hold time	$t_{HD;DAT}$		0	-	250	ns	Transmitting data
			0	-	- -	ns	Receiving data
SDA set-up time	$t_{SU;DAT}$		100	-	-	ns	
SCL/SDA rise time	t_R		-	-	300	ns	
SCL/SDA fall time	t_F		-	-	300	ns	
SDA valid time	$t_{VD;DAT}$		-	-	0.9	µs	
Set-up time for a repeated START condition	$t_{SU;STA}$		0.26	-	-	µs	
Set-up time for STOP condition	$t_{SU;STO}$		0.26	-	-	µs	
Capacitive load on bus line	CB		-	-	400	pF	
Low level input voltage	V_{IL}		0	-	$0.3xV_{DD}$	V	
High level input voltage	V_{IH}		$0.7xV_{DD}$	-	$1xV_{DD}$	V	
Low level output voltage	V_{OL}	33 mA sink current	-	-	0.4	V	

Table 20: Timing specifications for I2C communication, valid for T=-40°C … 125°C and VDD = VDDmin… VDDmax. The nomenclature above is according to the I2C Specification (UM10204, Rev. 6, April 4, 2014).

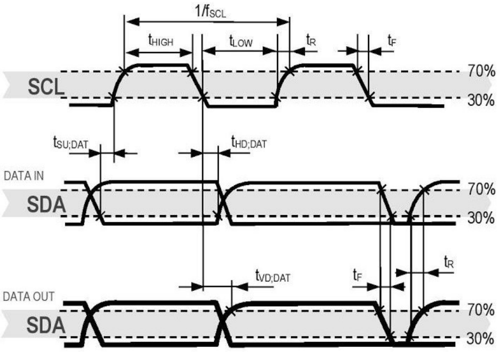


Figure 5: Timing diagram for digital input/output pads. SDA directions are seen from the sensor. Bold SDA lines are controlled by the sensor, plain SDA lines are controlled by the micro-controller. Note that SDA valid read time is triggered by falling edge of preceding toggle.

5 Packaging

The SHT85 is supplied in a single-in-line pin type package. The SHT35-DIS sensor housing consists of an epoxy-based mold compound, see “Datasheet SHT3x-DIS” for more information. The sensor opening of the housing is protected by a PTFE membrane dedicated to protect the sensor opening from liquids and dust according to IP67, see “Datasheet Membrane Option” for more information. The sensor head is connected to the pins by a small bridge to minimize heat conduction and response times. The pins are soldered to the FR4 substrate by lead-free solder paste. The gold plated backside of the sensor head is connected to the VSS pin. A 100nF capacitor is mounted on the front side between VDD and VSS. The device is fully RoHS compliant – thus it is free of of Pb, Cd, Hg, Cr(6+), PBB and PBDE. All pins are Au plated to avoid corrosion. They can be soldered or mate with most 1.27 mm (0.05’’) sockets, for example: Preci-dip / Mill-Max R851-83-004-20-001 or similar. When the sensor is further processed by soldering, it should be ensured that the solder connections between pins and the SHT85 PCB are not melted.

5.1 Traceability

The SHT85 provides a device specific serial number, which can be read-out via the serial interface (I2C), see the command in Table 21. The Serial number allows an unambiguous identification of each individual device.

Command	Hex Code
Get Serial Number	0x 36 82

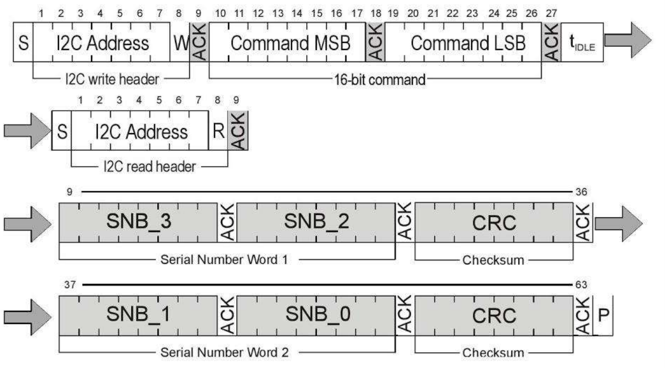


Table 21: Command to read out the Serial Number (Clear blocks are controlled by the microcontroller, grey blocks by the sensor.)

After issuing the measurement command and sending the ACK Bit the sensor needs the time t_{IDLE} = 0.5ms to respond to the I2C read header with an ACK Bit. Hence it is recommended to wait t_{IDLE} =0.5ms before issuing the read header. The Get Serial Number command returns 2 words; every word is followed by a CRC Checksum. Together the 2 words (SNB_3 to SNB_0 in Table 21, SNB_0 is the LSB, whereas SNB_3 is the MSB) constitute a unique serial number with a length of 32 bit. This serial number can be used to identify each sensor individually.

5.2 Package Outline

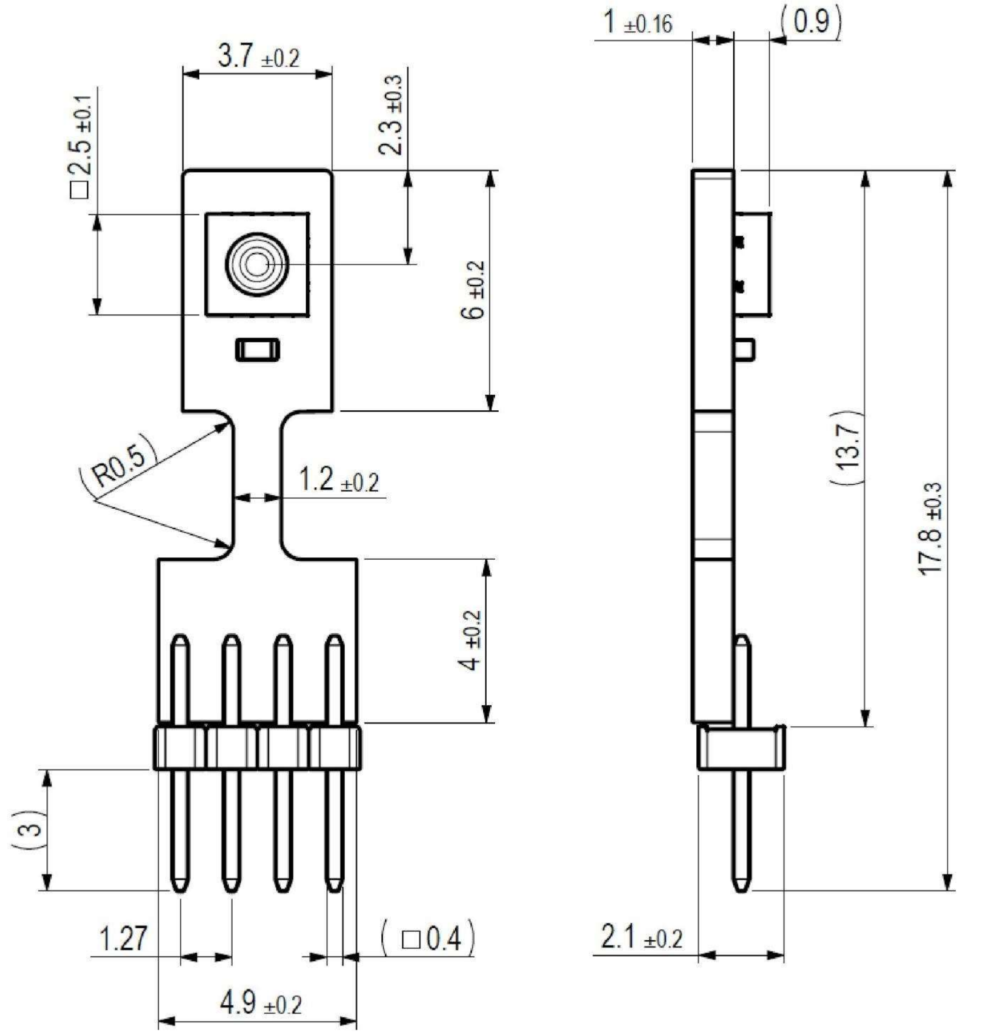
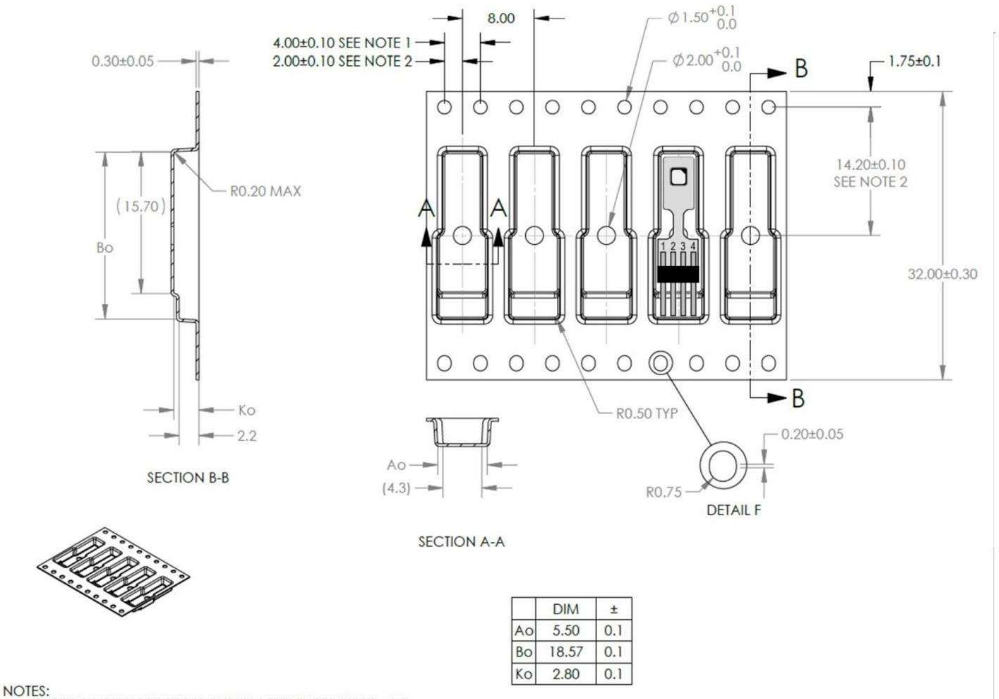


Figure 6: Dimensional drawing of the SHT85 sensor packaging. Dimensions are in mm (1mm = 0.039 inch).

6 Shipping Package

SHT85 are shipped in 32mm tape at 50pcs each. Dimensions of packaging tape are given in Figure 7. All tapes have a 10 pockets empty leader tape (first pockets of the tape) and a 10 pockets empty trailer tape (last pockets of the tape).



	DIM	±
Ao	5.50	0.1
Bo	18.57	0.1
Ko	2.80	0.1

NOTES:
1. 10 SPROCKET HOLE PITCH CUMULATIVE TOLERANCE ±0.2
2. POCKET POSITION RELATIVE TO SPROCKET HOLE MEASURED AS TRUE POSITION OF POCKET, NOT POCKET HOLE.
3. Ao AND Bo ARE MEASURED ON A PLANE AT A DISTANCE "R" ABOVE THE BOTTOM OF THE POCKET.

Figure 7 Tape configuration and unit orientation within tape, dimensions in mm (1mm = 0.039inch).

7 Quality

Qualification of the SHT85 is performed based on JEDEC guidelines. Furthermore, the SHT3x-DIS component qualification is based on the AEC Q 100 qualification test method.

7.1 Material Contents

The device is fully RoHS compliant, e.g. free of Pb, Cd, and Hg.

8 Ordering Information

The SHT85 can be ordered in tape and reel packaging, see Table 22. The reels are sealed into antistatic ESD bags.

Sensor Type	Packaging	Quantity	Order Number
SHT85	Tape Stripes	50	3.000.074

Table 22 SHT85 ordering information.

9 Further Information

For more in-depth information on the SHT85 and its application please consult the documents in Table 23. Parameter values specified in the datasheet overrule possibly conflicting statements given in references cited in this datasheet.

Document Name	Description	Source
SHT85 Shipping Package	Describes the standard shipping package	Available upon request.
Handling of SMD Packages Humidity Sensors	Assembly Guide	Available for download at the Sensirion humidity sensors download center: www.sensirion.com/humidity-download
Datasheet Humidity Sensor SHT3x Digital	All specifications of the SHT35-DIS	Available for download at the Sensirion humidity sensors download center: www.sensirion.com/humidity-download
Datasheet Humidity Sensor Filter Membrane SHT3x	All relevant specifications of the filter membrane	Available for download at the Sensirion humidity sensors download center: www.sensirion.com/humidity-download
Handling Instructions Humidity Sensors	Guidelines for proper handling of SHTxx humidity sensors	Available for download at the Sensirion humidity sensors download center: www.sensirion.com/humidity-download
Specification Statement Humidity Sensors	Definition of sensor specifications.	Available for download at the Sensirion humidity sensors download center: www.sensirion.com/humidity-download

Table 23 Documents containing further information relevant for the SHT85.

10 Important Notices

10.1 Warning, Personal Injury

Do not use this product as safety or emergency stop devices or in any other application where failure of the product could result in personal injury. Do not use this product for applications other than its intended and authorized use. Before installing, handling, using or servicing this product, please consult the data sheet and application notes. Failure to comply with these instructions could result in death or serious injury.

If the Buyer shall purchase or use SENSIRION products for any unintended or unauthorized application, Buyer shall defend, indemnify and hold harmless SENSIRION and its officers, employees, subsidiaries, affiliates and distributors against all claims, costs, damages and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if SENSIRION shall be allegedly negligent with respect to the design or the manufacture of the product.

10.2 ESD Precautions

The inherent design of this component causes it to be sensitive to electrostatic discharge (ESD). To prevent ESD-induced damage and/or degradation, take customary and statutory ESD precautions when handling this product.
See application note “ESD, Latchup and EMC” for more information.

10.3 Warranty

SENSIRION warrants solely to the original purchaser of this product for a period of 12 months (one year) from the date of delivery that this product shall be of the quality, material and workmanship defined in SENSIRION’s published specifications of the product. Within such period, if proven to be defective, SENSIRION shall repair and/or replace this product, in SENSIRION’s discretion, free of charge to the Buyer, provided that:

- notice in writing describing the defects shall be given to SENSIRION within fourteen (14) days after their appearance;
- such defects shall be found, to SENSIRION’s reasonable satisfaction, to have arisen from SENSIRION’s faulty design, material, or workmanship;
- the defective product shall be returned to SENSIRION’s factory at the Buyer’s expense; and
- the warranty period for any repaired or replaced product shall be limited to the unexpired portion of the original period.

This warranty does not apply to any equipment which has not been installed and used within the specifications recommended by SENSIRION for the intended and proper use of the equipment. EXCEPT FOR THE WARRANTIES EXPRESSLY SET FORTH HEREIN, SENSIRION MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THE PRODUCT. ANY AND ALL WARRANTIES, INCLUDING WITHOUT LIMITATION, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, ARE EXPRESSLY EXCLUDED AND DECLINED.

SENSIRION is only liable for defects of this product arising under the conditions of operation provided for in the data sheet and proper use of the goods. SENSIRION explicitly disclaims all warranties, express or implied, for any period during which the goods are operated or stored not in accordance with the technical specifications.

SENSIRION does not assume any liability arising out of any application or use of any product or circuit and specifically disclaims any and all liability, including without limitation consequential or incidental damages. All operating parameters, including without limitation recommended parameters, must be validated for each customer’s applications by customer’s technical experts. Recommended parameters can and do vary in different applications.

SENSIRION reserves the right, without further notice, (i) to change the product specifications and/or the information in this document and (ii) to improve reliability, functions and design of this product.

11 Revision History

Release Date	Version	Page(s)	Changes
06. November 2018	1.0	All	Initial Release

12 Headquarters and Subsidiaries

SENSIRION AG
Laubisruetistr. 50
CH-8712 Staefa ZH
Switzerland

phone: +41 44 306 40 00
fax: +41 44 306 40 30
info@sensirion.com
www.sensirion.com

Sensirion Taiwan Co. Ltd.
phone: +41 44 306 40 00
info@sensirion.com

Sensirion Inc. USA
phone: +1 312 690 5858
info-us@sensirion.com
www.sensirion.com

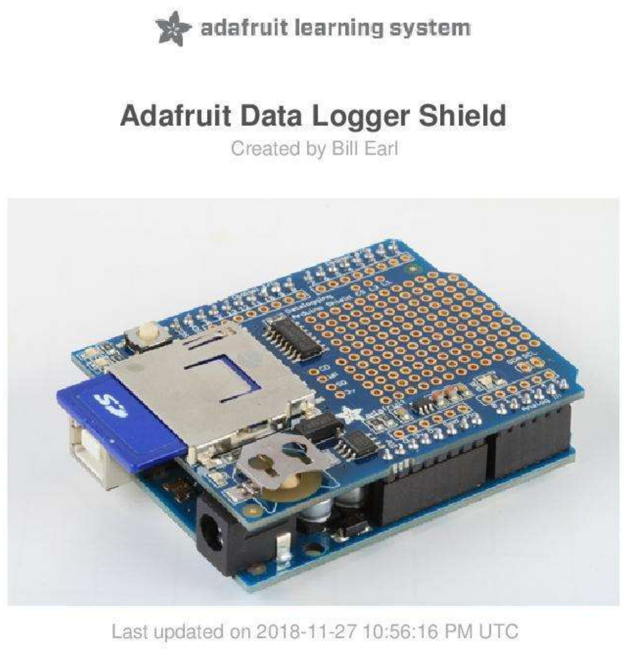
Sensirion Japan Co. Ltd.
phone: +81 3 3444 4940
info-jp@sensirion.com
www.sensirion.co.jp

Sensirion Korea Co. Ltd.
phone: +82 31 337 7700~3
info-kr@sensirion.com
www.sensirion.co.kr

Sensirion China Co. Ltd.
phone: +86 755 8252 1501
info-cn@sensirion.com
www.sensirion.com.cn/

To find your local representative, please visit www.sensirion.com/contact

10.8 Appendix H: Adafruit Data Logging Shield Datasheet

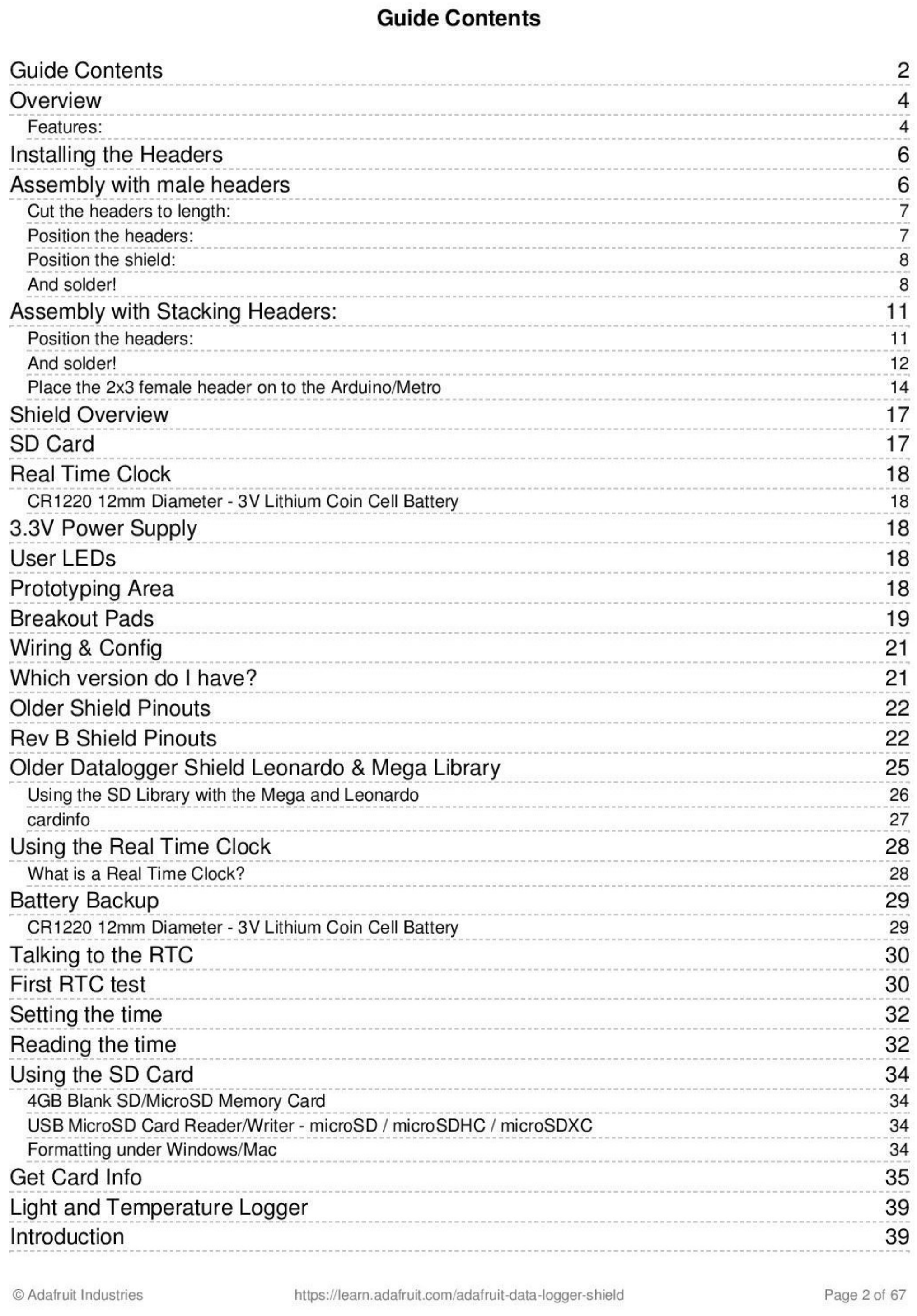


adafruit learning system

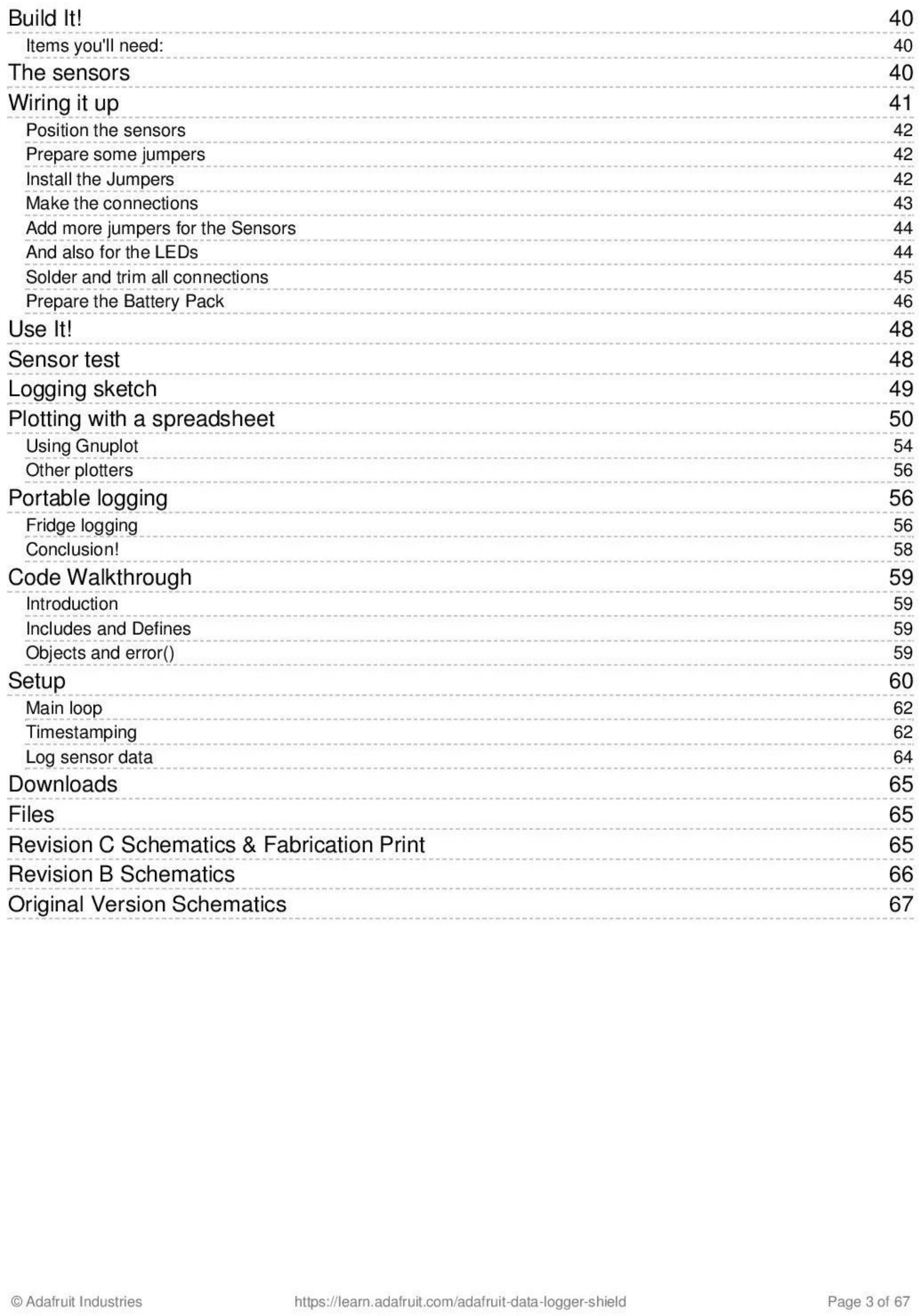
Adafruit Data Logger Shield

Created by Bill Earl

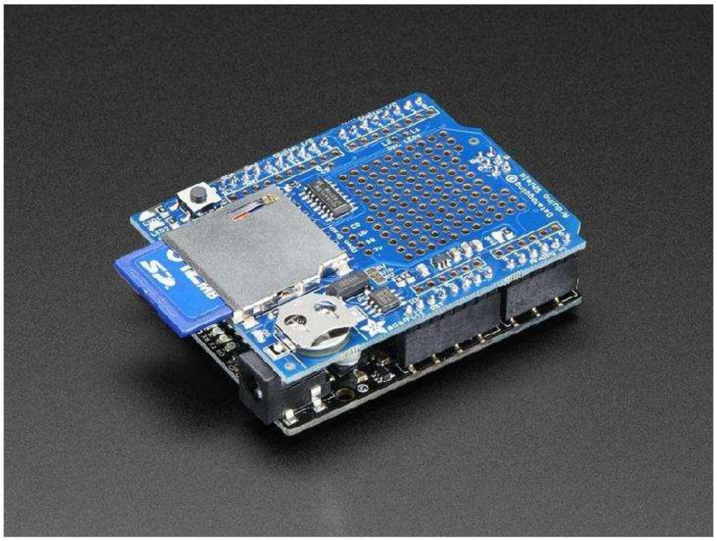
Last updated on 2018-11-27 10:56:16 PM UTC



Guide Contents



Overview



Here's a handy Arduino shield: we've had a lot of people looking for a dedicated and well-designed data logging shield. We worked hard to engineer an inexpensive but well-rounded design. This shield makes it easy to add a 'hard disk' with gigabytes of storage to your Arduino!

Our latest version of this popular shield has all the features of the popular original, and is "R3" compatible so you can use it with just about any Arduino or compatible. You can be up and running with it in less than 15 minutes - saving data to files on any FAT16 or FAT32 formatted SD card, to be read by any plotting, spreadsheet or analysis program. This tutorial will also show you how to use two free software programs to plot your data. The included RTC (Real Time Clock) can be used to timestamp all your data with the current time, so that you know precisely what happened when!

The data logger is a reliable, well-rounded and versatile design. It is easily expanded or modified and come well supported with online documentation and libraries

Features:

- SD card interface works with FAT16 or FAT32 formatted cards. Built in 3.3v level shifter circuitry lets you read or write super fast and prevents damage to your SD card
- Real time clock (RTC) keeps the time going even when the Arduino is unplugged. The coin cell battery backup lasts for years
- Included libraries and example code for both SD and RTC mean you can get going quickly
- Prototyping area for soldering connectors, circuitry or sensors.
- Two configurable indicator LEDs
- Onboard 3.3v regulator is both a reliable reference voltage and also reliably runs SD cards that require a lot of power to run
- Uses the "R3 layout" I2C and ICSP/SPI ports so it is compatible with a wide variety of Arduinos and Arduino-compatibles

With this new version you can use it with:

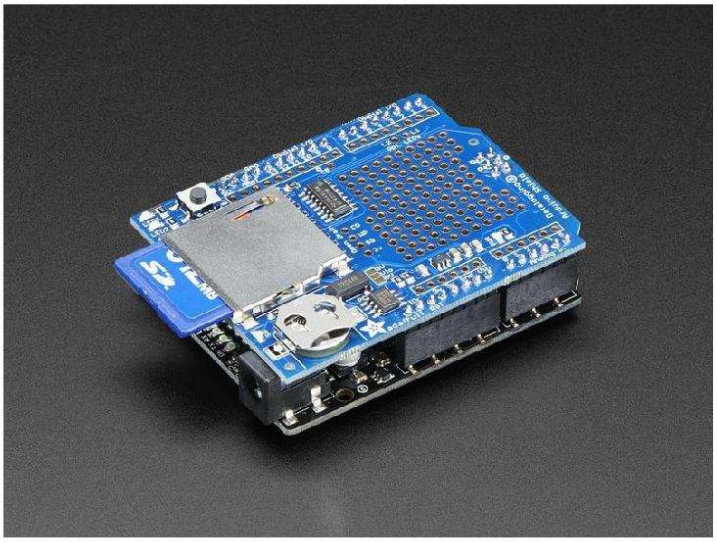
- Arduino UNO or ATmega328 compatible - 4 analog channels at 10 bit resolution, 6 if RTC is not used
- Arduino Leonardo or ATmega32u4 compatible - 12 analog channels at 10 bit resolution
- Arduino Mega or ATmega2560 compatible - 16 analog inputs (10-bit)
- Arduino Zero or ATSAMD21 compatible - 6 analog inputs (12-bit)
- Arduino Due compatible - 12 analog inputs (12-bit)

Of course you can log anything you like, including digital sensors that have Arduino libraries, serial data, bit timings, and more!

Installing the Headers

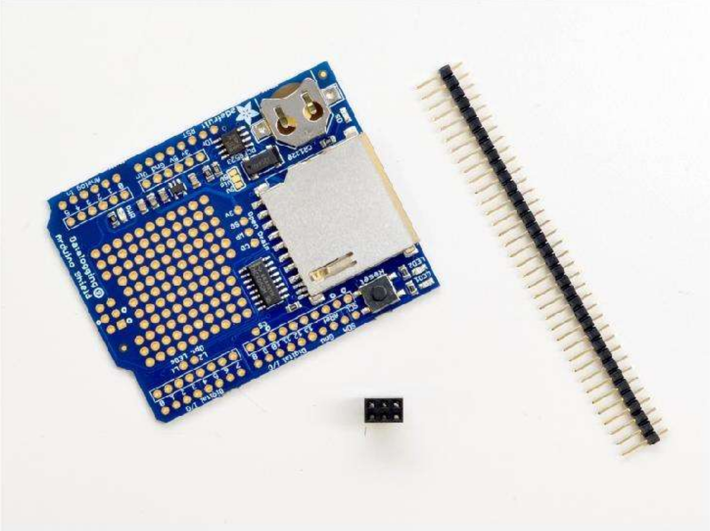
The Adafruit Data Logger shield comes tested assembled with all components and SD socket already on it, but you'll still need need to put headers on so you can plug it into an Arduino

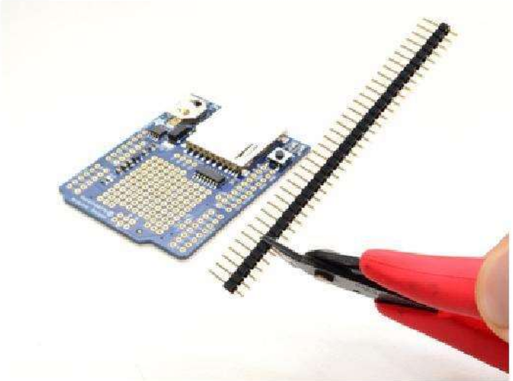
We don't pre-assemble the headers on because there's **two** options! You can either use plain 0.1" male headers (included with the shield) or Arduino Shield Stacking headers (http://adafru.it/85). Both options additionally **require** a 2x3 female header soldered on.



Assembly with male headers

Most people will be happy with assembling he shield with male headers. The nice thing about using these is they don't add anything to the height of the project, and they make a nice solid connection. However, you won't be able to stack another shield on top. Trade offs!





Cut the headers to length:

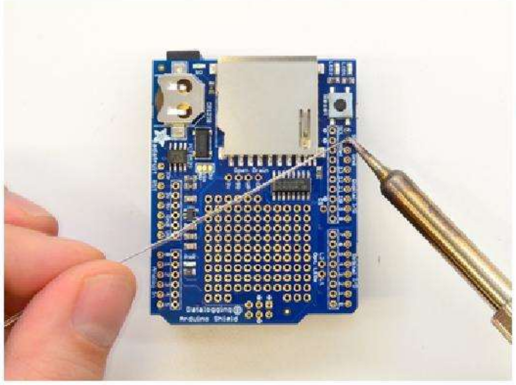
Line the header strip up with the holes on the edge of the shield and cut 4 sections of header strip to fit.

Position the headers:

Insert the header sections - long pins down - into the female headers on your Arduino/Metro. Additionally insert the 2x3 female header into the corresponding pins on the opposite side as the USB.

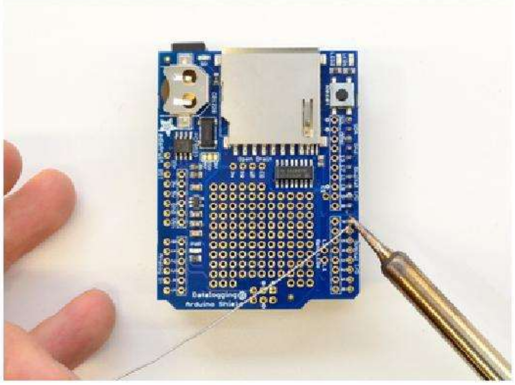
Position the shield:

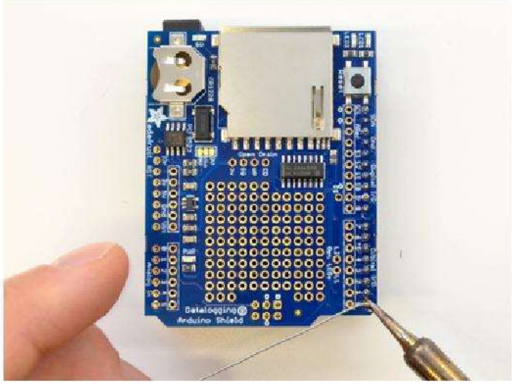
Align the shield with the header pins and press down.



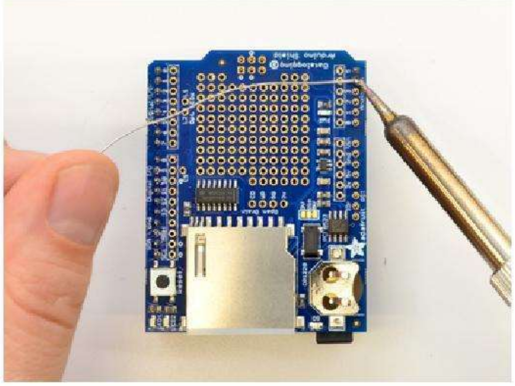
And solder!

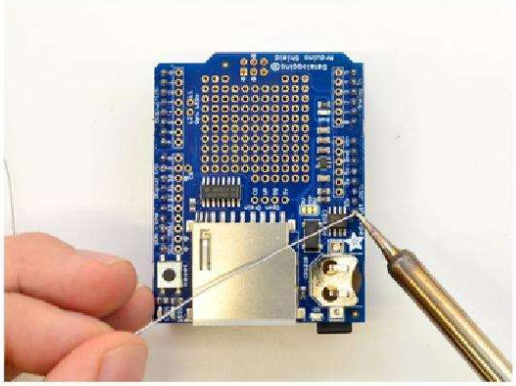
Solder each pin to assure good electrical contact. For tips on soldering, refer to the Adafruit Guide to Excellent Soldering (https://adafru.it/c6b).

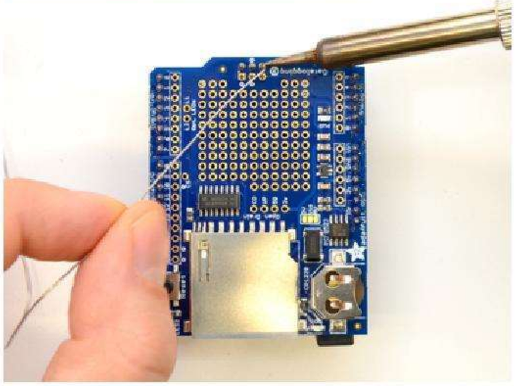


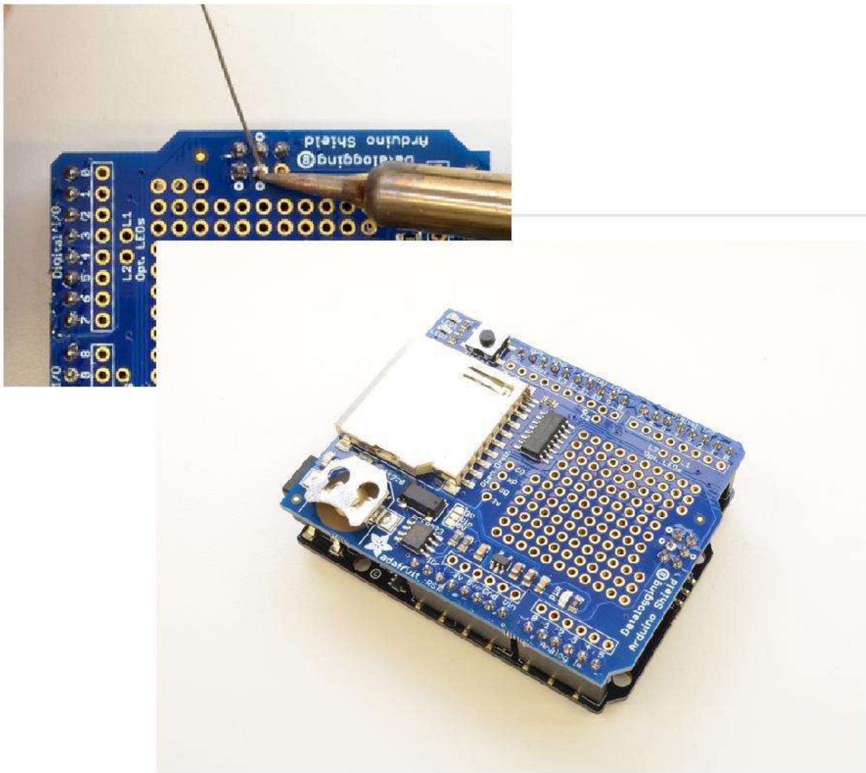


Flip around and solder the other side as well as the 2x3 header









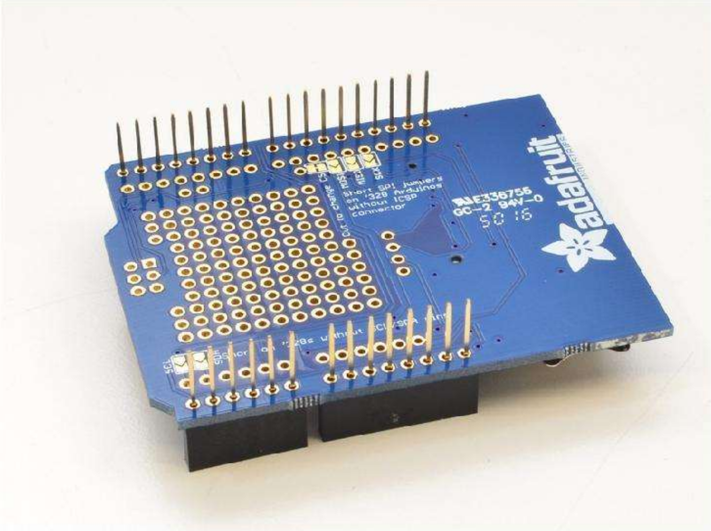
Assembly with Stacking Headers:

Stacking headers give your data logger shield extra flexibility. You can combine it with other shields such as the RGB/LCD Display shield (http://adafru.it/714) to make a compact logging instrument complete with a user interface. You can also stack it with one or more Proto-Shields (http://adafru.it/51) to add even more prototyping space for interfacing to sensors.

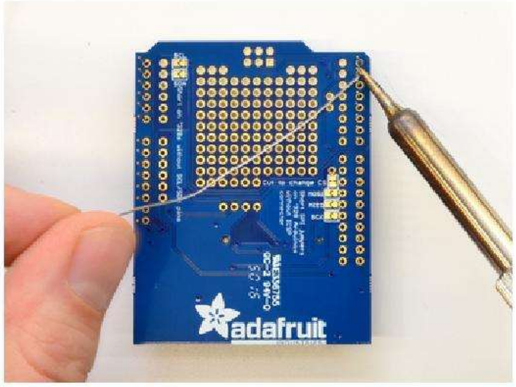
Stacking headers are installed from the top of the board instead of the bottom, so the procedure is a little different than for installing simple male headers.

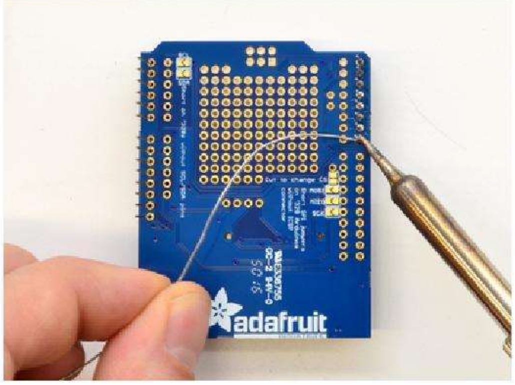
Position the headers:

Insert the headers **from the top** of the shield, then **flip the shield over** and place it on a flat surface. Straighten the headers so that they are vertical.



Be sure to insert the headers from the TOP of the shield so that they can be soldered from the BOTTOM.



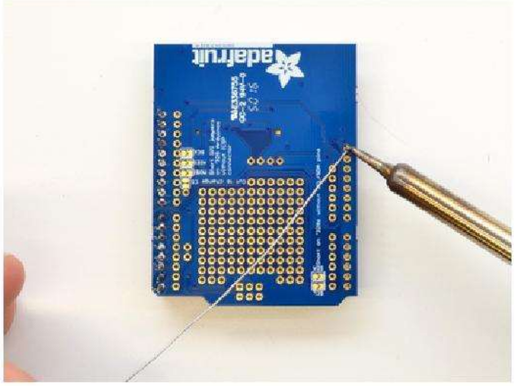


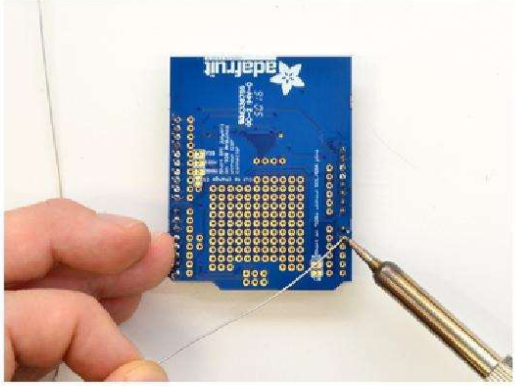
And solder!

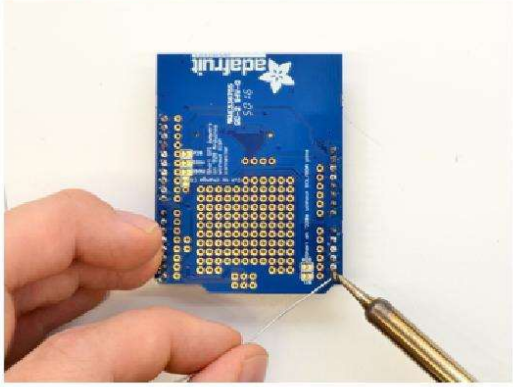
Solder each pin for a solid electrical connection.

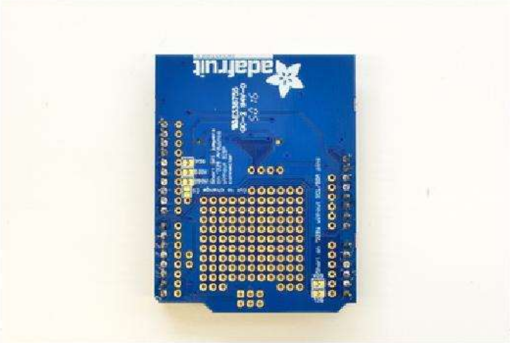
Tip: Solder one pin from each header section. If any of them are crooked, simply re-heat the one solder joint and straighten it by hand. Once all headers are straight, continue soldering the rest of the pins.

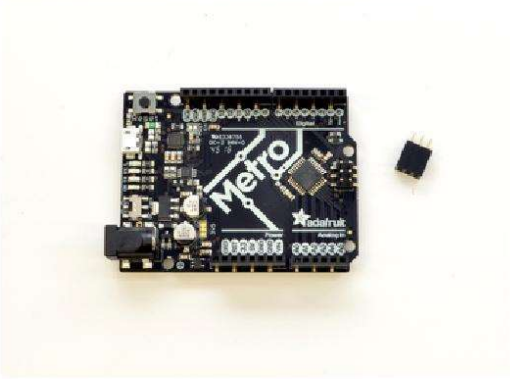
Flip and solder the other side



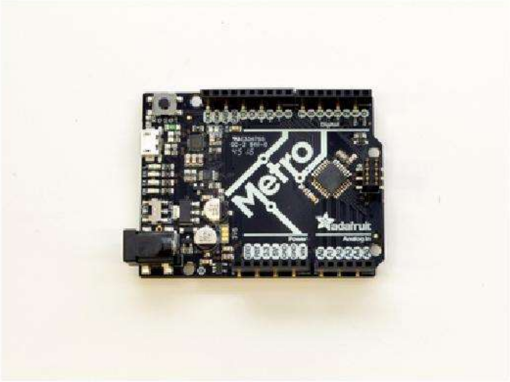


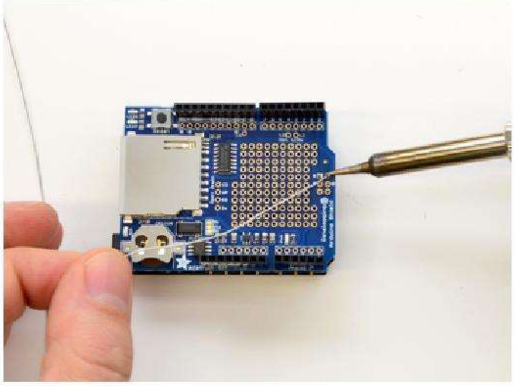


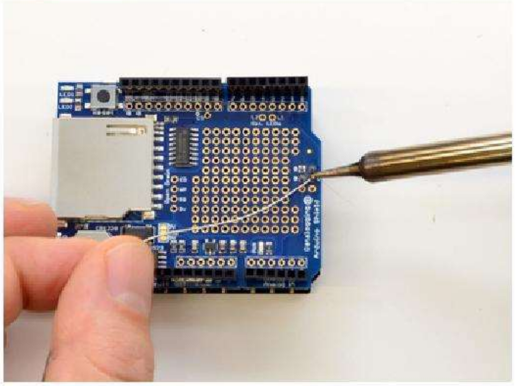




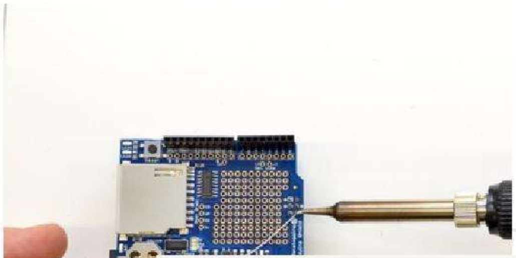
Place the 2x3 female header on to the Arduino/Metro

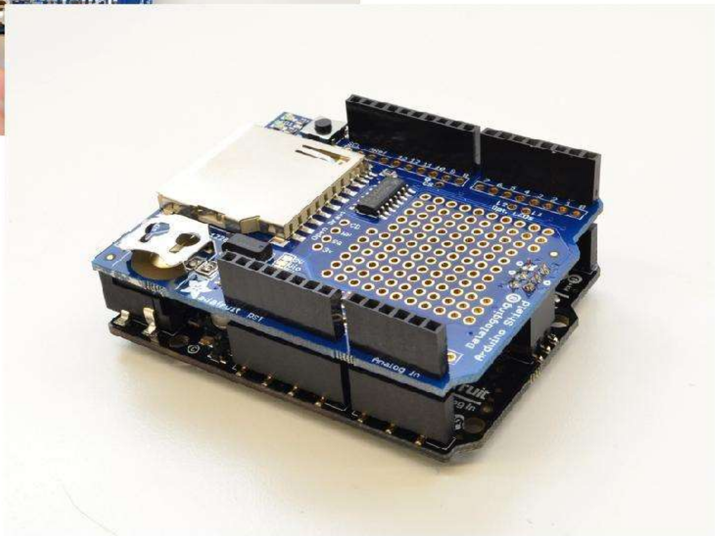






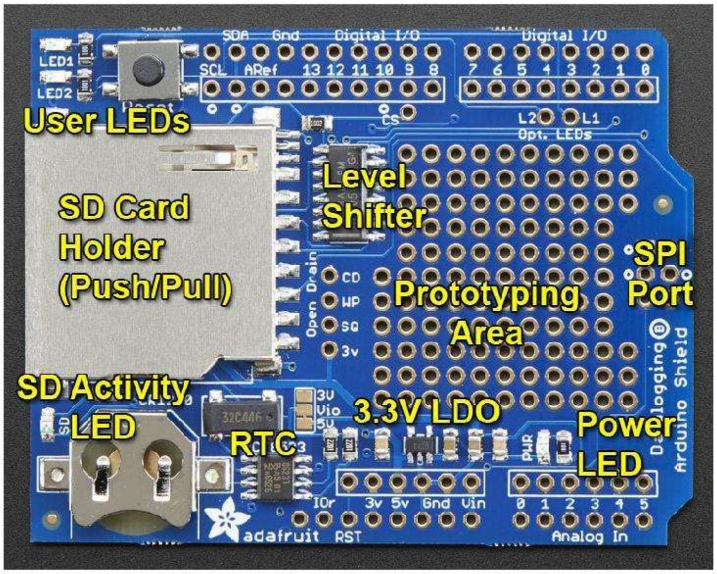
Place the board on the Metro and solder the 2x3 header



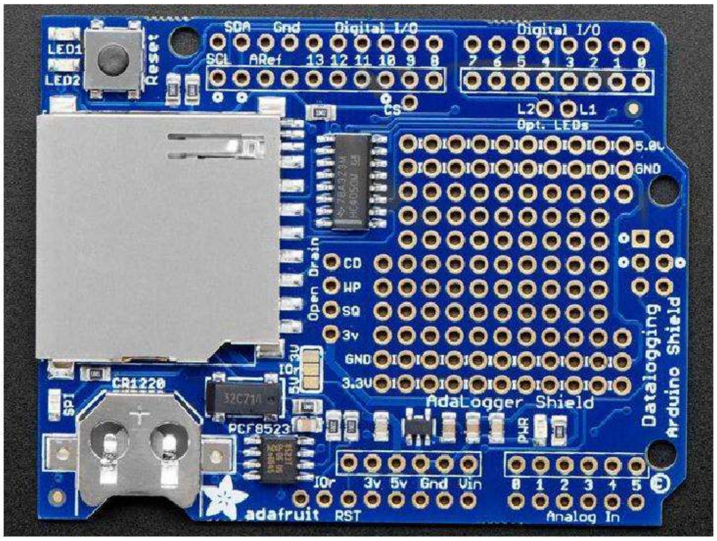

Datalogging Arduino Shield
adafruit
RST
Analog In

Shield Overview

The datalogger shield has a few things to make it an excellent way to track data. Here's a rough map of th shield:



Our latest version adds power rails for 5V, 3.3V and Ground:



SD Card

The big SD card holder can fit any SD/MMC storage up to 32G and and small as 32MB (Anything formatted FAT16 or FAT32) If you have a MicroSD card, there are low cost adapters which will let you fit these in. SD cards are tougher to lose than MicroSD, and there's plenty of space for a full size holder.

Simply Push to insert, or Pull to remove the card from this slot

The **SD Activity LED** is connected to the clock pin, it will blink when data goes over SPI, which can help you detect when its ok to remove or insert the SD card or power down the Arduino.

The **Level Shifter** moves all signals from 3.3 or 5V down to 3.3V so you can use this shield with *any* Arduino safely and not damage cards. Cheaper shields use resistors to level shift, but this doesn't work well at high speed or at all voltage levels!

Real Time Clock

This is the time-keeping device. It includes the 8-pin chip, the rectangular 32KHz crystal and a battery holder

The battery holder must contain a battery in order for the RTC to keep track of time when power is removed from the Arduino! Use any CR1220 compatible coin cell

CR1220 12mm Diameter - 3V Lithium Coin Cell Battery

$0.95
IN STOCK

ADD TO CART

3.3V Power Supply

An on-board 3.3V LDO (low drop-out type) regulator keeps the shield's 3V parts running smoothly. Some old Arduinos did not have a full 3.3V regulator and writing to an SD card could cause the Arduino to reboot. To maintain compatibility we just keep it there. There's also a green PWR (Power) good LED to the right

User LEDs

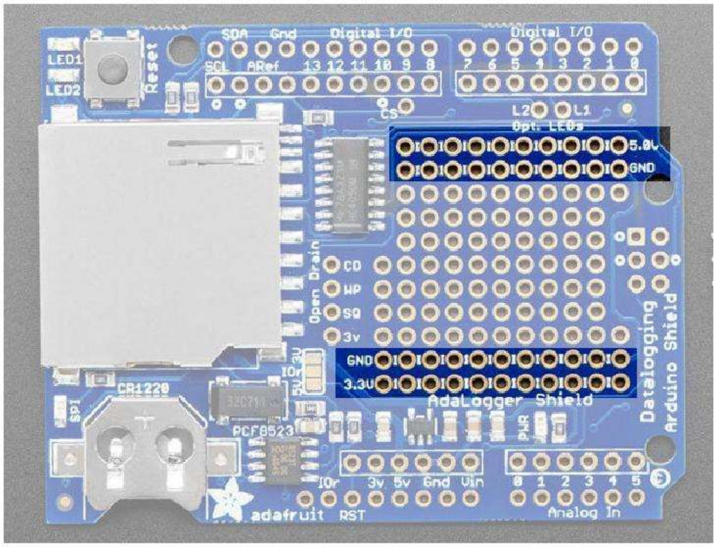
We have two user-configuratble LEDs. Connect a wire from any Arduino pin to **L1** or **L2** marked pads and pull high to turn on **LED1** or **LED2**

The reset button to the right of the LEDs, will reset the entire Arduino, handy for when you want to restart the board

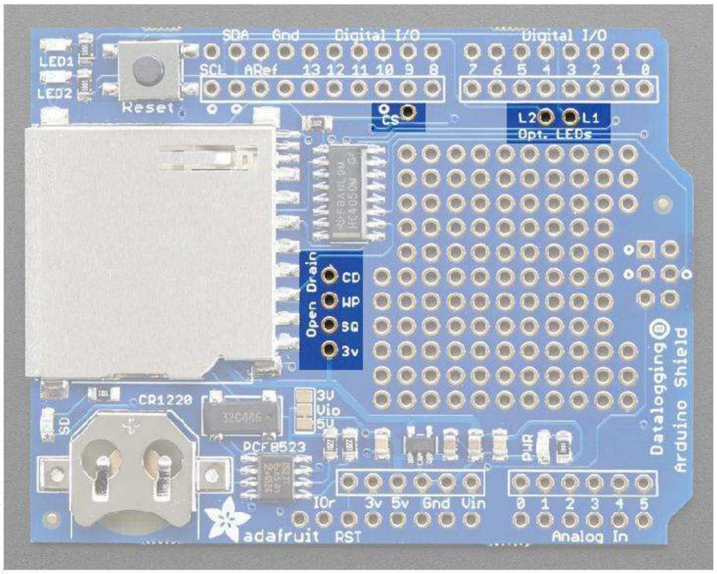
Prototyping Area

The big middle section is filled with 0.1" grid prototyping holes so you can customize your shield with sensors or other circuitry.

The top two and bottom two rows of proto holes are power rails.



Breakout Pads



We also have some extra breakouts shown above, around the breakout board area.

To the right of the SD card holder:

- **CD** - this is the card detect pad on the SD card. When this is connected to ground, an SD card is inserted. It is open-drain, use a pullup (either physical resistor or enabled in software)

- **WP** - this is the Write Protect pad on the SD card, you can use this to detect if the write-protect tab is on the card by checking this pin. It is open-drain, use a pullup (either physical resistor or enabled in software)
- **SQ** - this is the optional Squarewave output from the RTC. You have to send the command to turn this on but its a way of optionally getting a precision squarewave. We use it primarily for testing. The output is open drain so a pullup (either physical resistor or enabled in software)
- **3V** - this is the 3V out of the regulator. Its a good quality 3.3V reference which you may want to power sensors. Up to 50mA is available

Near Digital #10

- **CS** - this is the **Chip Select** pin for the SD card. If you need to cut the trace to pin 10 because it is conflicting, this pad can be soldered to any digital pin and the software re-uploaded

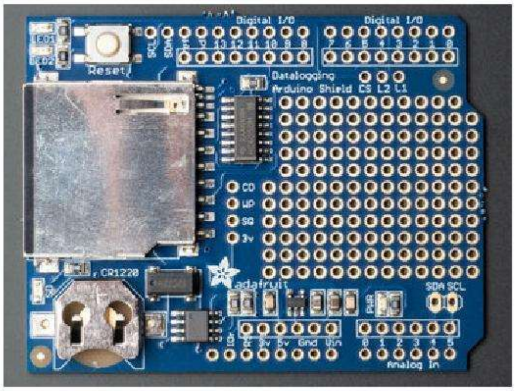
Near Digital #3 and #4

- **L2** and **L1** - these are optional user-LEDs. Connect to any digital pin, pull high to turn on the corresponding LED. The LEDs already have 470 ohm resistors in series.

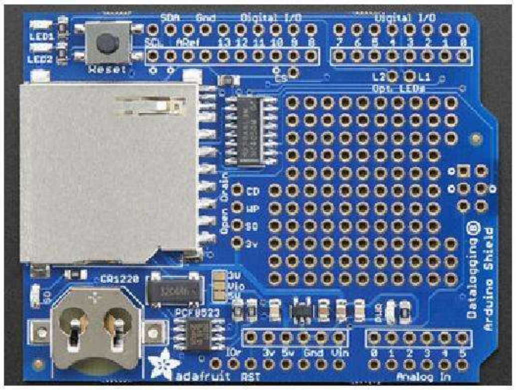
Wiring & Config

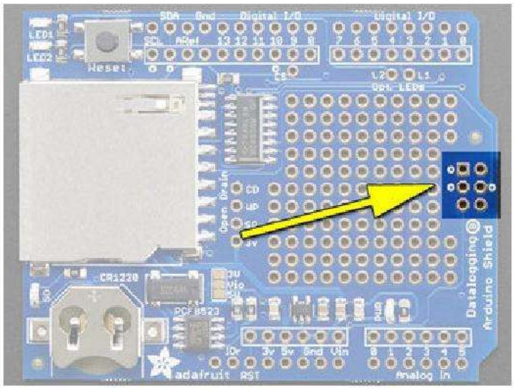
As of revision B of the Datalogger shield, we've moved away from using digital pins 10, 11, 12, 13 for SPI and A4, A5 for I2C. We now use the 2x3 ICSP header, which means that you don't need special customized I2C or SPI libraries to use with Mega or Leonardo or Zero (or any other future type) of Arduino!

Which version do I have?



This is the older Datalogger shield. In particular, note that the prototyping area is completely full of 0.1" spaced holes





This is the "R3 compatible" Datalogger. Note that it has a smaller prototyping area and that there is a 2x3 SPI header spot on the right

Older Shield Pinouts

On the older shields, the pinout was *fixed* to be:

- **Digital #13** - SPI clock
- **Digital #12** - SPI MISO
- **Digital #11** - SPI MOSI
- **Digital #10** - SD Card chip select (can cut a trace to re-assign)
- **SDA** connected to **A4**
- **SCL** connected to **A5**

The RTC (DS1307) I2C logic level was fixed to 5V

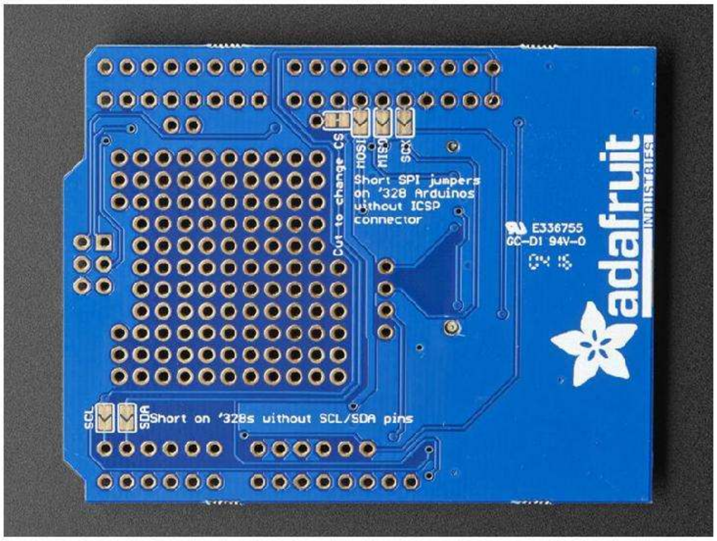
Rev B Shield Pinouts

- **ICSP SCK** - SPI clock
- **ICSP MISO** - SPI MISO
- **ICSP MOSI** - SPI MOSI
- **Digital #10** - SD Card chip select (can cut a trace to re-assign)
- **SDA** *not* connected to **A4**
- **SCL** *not* connected to **A5**

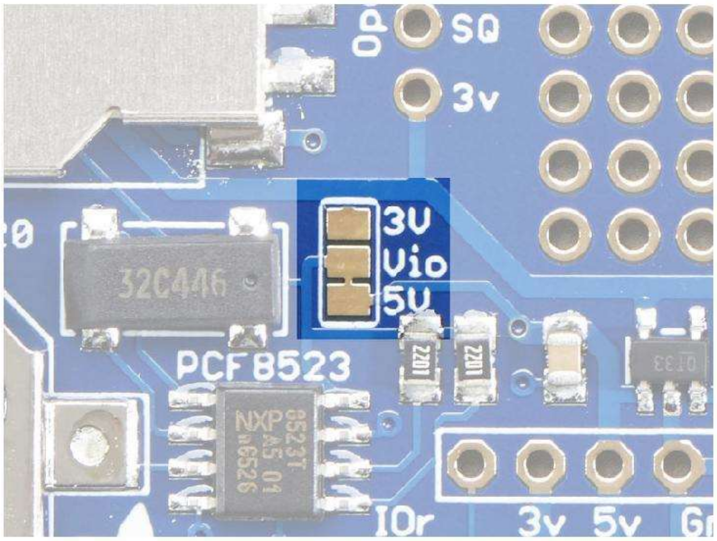
The RTC (PCF8523) logic level can be 3V or 5V

On an UNO, note that Digital #13 is the same as ICSP SCK, #12 is ICSP MISO, #11 is ICSP MOSI, SDA is tied to A4 and SCL is A5. However, that is only true on the UNO! Other Arduino's have different connections. Since the shield no longer makes the assumption it's on an UNO, it is the most cross-compatible shield.

On the bottom of the Rev B shield, you can see that if you have an older Arduino where there is no ICSP 2x3 header, and no SDA/SCL pins, you can short the solder jumpers closed.


Cut to change CS
MOSI
MISO
SCK
Short SPI jumpers on '328 Arduinos without ICSP connector
E336755
GC-D1 94V-0
adafruit
INDUSTRIES
SCL
SDA
Short on '328s without SCL/SDA pins

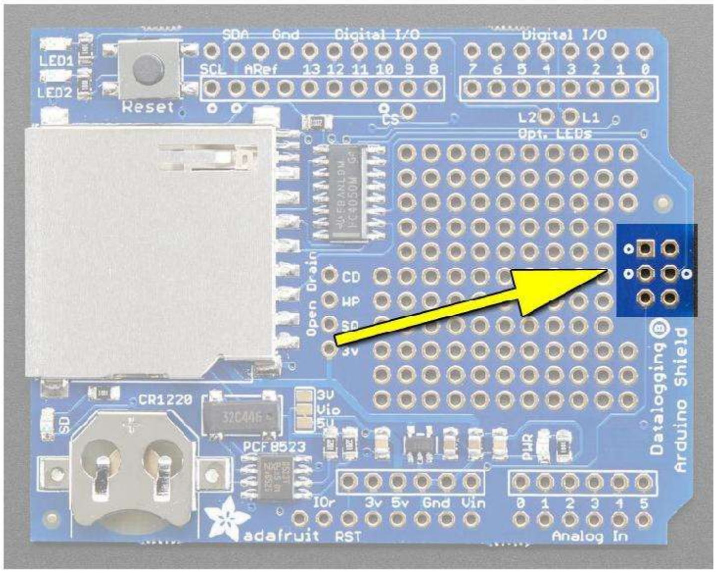
If you are using the shield with a 3.3V logic Arduino, you may want to change the **Vio** jumper. This is what the 10K pullups for I2C are pulled up to. Honestly, the pullups are very weak so if you forget, it's not a big deal. But if you can, cut the small trace between the center pad and 5V and solder the other side so that Vio is connected to 3V


SQ
3v
3V
Vio
5V
32C446
PCF8523
NXP
8523T
A5 01
6526
IOr
3v
5v

Older Datalogger Shield Leonardo & Mega Library

This is ONLY required if you have the older Datalogger shield which does not have the SPI port connection.

This is ONLY required if you are using a Leonardo or Mega with the older Datalogger shield!

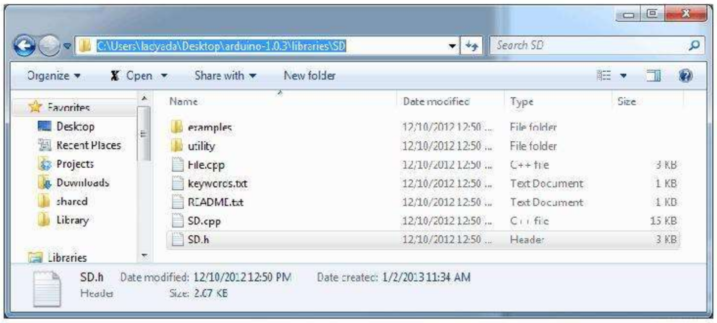


If your shield looks like the above, and has the 2x3 pin header on the right, *skip this page!*

If your shield does not have the 2x3 pin header section **and** you are using a Mega or Leonardo (e.g. not UNO-compatible) then you can keep reading!

If you are using an Leonardo or Mega with the older datalogging shield, you will have to replace the existing SD card library to add 'SD card on any pin' support. **If you have an Uno/Duemilanove/Diecimila, this is not required. If you have a rev B shield, this is also not required!**

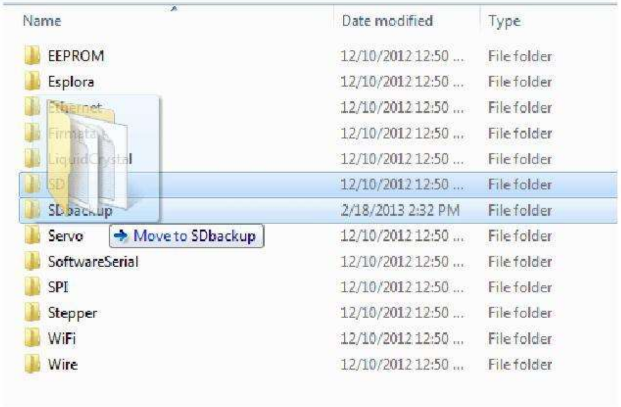
First, find the "core libraries" folder - if you are using Windows or Linux, it will be in the folder that contains the **Arduino** executable, look for a **libraries** folder. Inside you will see an **SD** folder (inside that will be **SD.cpp SD.h** etc)



Name	Date modified	Type	Size
examples	12/10/2012 12:50 ...	File folder	
utility	12/10/2012 12:50 ...	File folder	
File.cpp	12/10/2012 12:50 ...	C++ file	3 KB
keywords.txt	12/10/2012 12:50 ...	Text Document	1 KB
README.txt	12/10/2012 12:50 ...	Text Document	1 KB
SD.cpp	12/10/2012 12:50 ...	C++ file	15 KB
SD.h	12/10/2012 12:50 ...	Header	3 KB

SD.h Date modified: 12/10/2012 12:50 PM Date created: 1/2/2013 11:34 AM
Header Size: 2.67 KB

Ourside the **libraries** folder, make a new folder called **SDbackup**. Then drag the **SD**folder into **SDbackup**, this will 'hide' the old **SD** library without deleting it. *Note that SDBackup must be **outside** of the libraries folder in order to effectively 'hide' the SD library.*



Name	Date modified	Type
EEPROM	12/10/2012 12:50 ...	File folder
Esplora	12/10/2012 12:50 ...	File folder
Ethernet	12/10/2012 12:50 ...	File folder
Firmata	12/10/2012 12:50 ...	File folder
LiquidCrystal	12/10/2012 12:50 ...	File folder
SD	12/10/2012 12:50 ...	File folder
SDbackup	2/18/2013 2:32 PM	File folder
Servo	12/10/2012 12:50 ...	File folder
SoftwareSerial	12/10/2012 12:50 ...	File folder
SPI	12/10/2012 12:50 ...	File folder
Stepper	12/10/2012 12:50 ...	File folder
WiFi	12/10/2012 12:50 ...	File folder
Wire	12/10/2012 12:50 ...	File folder

Now we'll grab the new SD library, visit https://github.com/adafruit/SD (https://adafru.it/aP6) and click the**ZIP** download button, or click the button below

https://adafru.it/cxl

https://adafru.it/cxl

Uncompress and rename the uncompressed folder **SD**. Check that the **SD** folder contains **SD.cpp** and **SD.h**

Place the **SD** library folder your sketchbook libraries folder. You may need to create the libraries subfolder if its your first library. For more details on how to install libraries, check out our ultra-detailed tutorial at (https://adafru.it/aYM)http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use (https://adafru.it/aYM)

Using the SD Library with the Mega and Leonardo

Because the Mega and Leonardo do not have the same hardware SPI pinout, you need to specify which pins you will be using for SPI communication with the card. For the data logger shield, these will be pins 10, 11, 12 and 13. Find the location in your sketch where SD.begin() is called (like this):

```
// see if the card is present and can be initialized:
if (!SD.begin(chipSelect)) {
```

and change it to add these pin numbers as follows:

```
// see if the card is present and can be initialized:
if (!SD.begin(10, 11, 12, 13)) {
```

cardinfo

The cardinfo sketch uses a lower level library to talk directly to the card, so it calls card.init() instead of SD.begin().

```
// we'll use the initialization code from the utility libraries
// since we're just testing if the card is working!
while (!card.init(SPI_HALF_SPEED, chipSelect)) {
```

When calling card.init(), you must change the call to specify the SPI pins, as follows:

```
// we'll use the initialization code from the utility libraries
// since we're just testing if the card is working!
while (!card.init(SPI_HALF_SPEED, 10, 11, 12, 13)) {
```

Using the Real Time Clock

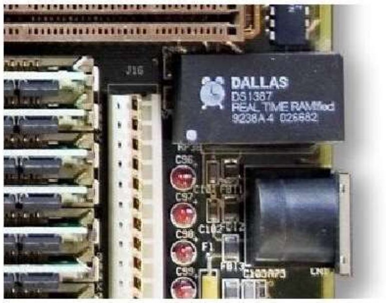
What is a Real Time Clock?

When logging data, it's often really really useful to have timestamps! That way you can take data one minute apart (by checking the clock) or noting at what time of day the data was logged.

The Arduino does have a built-in timekeeper called **millis()** and theres also timers built into the chip that can keep track of longer time periods like minutes or days. So why would you want to have a separate RTC chip? Well, the biggest reason is that **millis()** only keeps track of time *since the Arduino was last powered* - that means that when the power is turned on, the millisecond timer is set back to 0. The Arduino doesnt know its 'Tuesday' or 'March 8th' all it can tell is 'Its been 14,000 milliseconds since I was last turned on'.

OK so what if you wanted to set the time on the Arduino? You'd have to program in the date and time and you could have it count from that point on. But if it lost power, you'd have to reset the time. Much like very cheap alarm clocks: every time they lose power they blink **12:00**

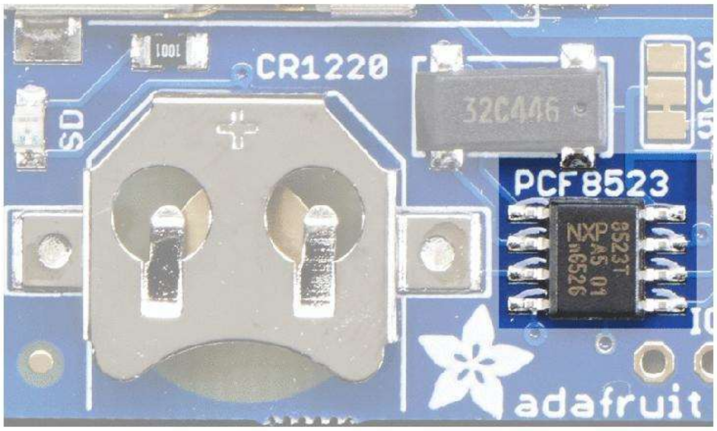
While this sort of basic timekeeping is OK for some projects, a data-logger will need to have **consistent timekeeping that doesnt reset when the Arduino battery dies or is reprogrammed**. Thus, we include a separate RTC! The RTC chip is a specialized chip that just keeps track of time. It can count leap-years and knows how many days are in a month, but it doesn't take care of Daylight Savings Time (because it changes from place to place)



This image shows a computer motherboard with a Real Time Clock called the DS1387 (https://adafru.it/aX0). Theres a lithium battery in there which is why it's so big.

The RTC we'll be using is the *PCF8523 (https://adafru.it/reb)* or the *DS1307* (https://adafru.it/rec).

If you have an Adafruit Datalogger Shield rev B, you will be using the PCF8523 - this RTC is newer and better than the DS1307. Look on your shield to see if you see **PCF8523** written above the chip.



If you have an older Datalogger shield, you will be using the DS1307 - there's no text so you'll just need to remember that if it *doesn't* say PCF8523 it's the DS1307

Battery Backup

As long as it has a coin cell to run it, the RTC will merrily tick along for a long time, even when the Arduino loses power, or is reprogrammed.

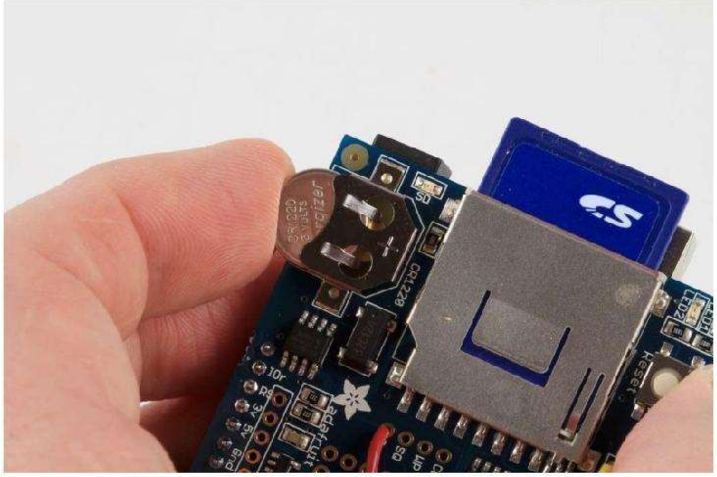
Use any CR1220 3V lithium metal coin cell battery:

CR1220 12mm Diameter - 3V Lithium Coin Cell Battery

$0.95
IN STOCK

ADD TO CART

You MUST have a coin cell installed for the RTC to work, if there is no coin cell, it will act strangely and possibly hang the Arduino when you try to use it, so ALWAYS make SURE there's a battery installed, even if it's a dead battery.



Talking to the RTC

The RTC is an i2c device, which means it uses 2 wires to to communicate. These two wires are used to set the time and retrieve it. On the Arduino UNO, these pins are also wired to the **Analog 4** and **5** pins. This is a bit annoying since of course we want to have up to 6 analog inputs to read data and now we've lost two.

For the RTC library, we'll be using a fork of JeeLab's excellent RTC library, which is available on GitHub (https://adafru.it/c7r). You can do that by visiting the github repo and manually downloading or, easier, just click this button to download the zip

https://adafru.it/cxm

https://adafru.it/cxm

Rename the uncompressed folder **RTClib** and check that the **RTClib** folder contains **RTClib.cpp** and **RTClib.h**

Place the **RTClib** library folder your **arduinosketchfolder/libraries/** folder.
You may need to create the **libraries** subfolder if it's your first library. Restart the IDE.

We also have a great tutorial on Arduino library installation at:
http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use

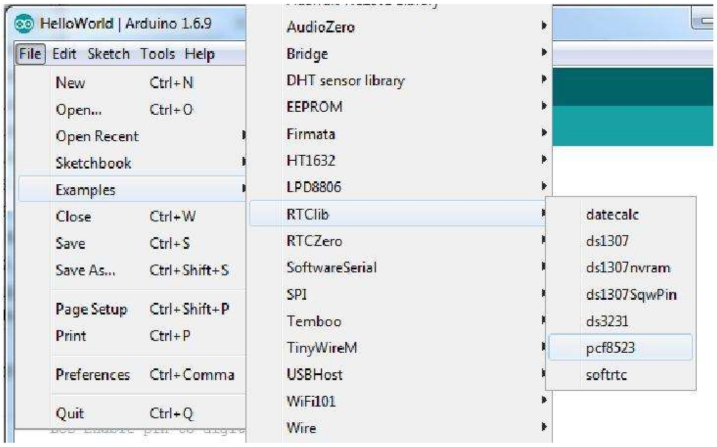
Once done, restart the IDE

First RTC test

The first thing we'll demonstrate is a test sketch that will read the time from the RTC once a second. We'll also show what happens if you remove the battery and replace it since that causes the RTC to halt. So to start, remove the battery from the holder while the Arduino is not powered or plugged into USB. Wait 3 seconds and then replace the battery. This resets the RTC chip. Now load up the matching sketch for your RTC

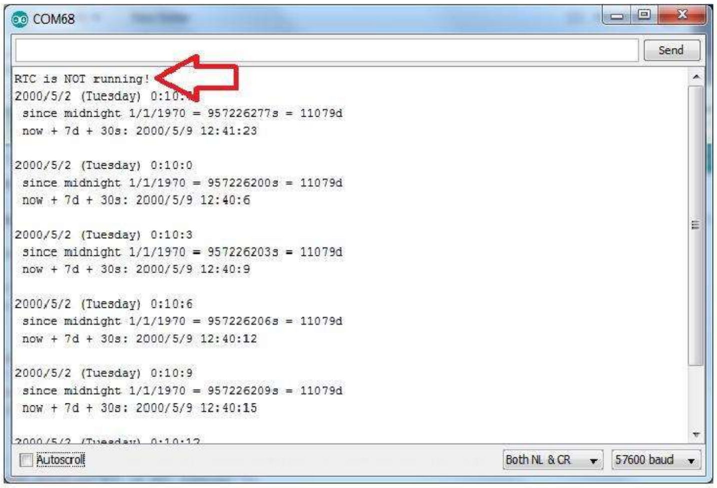
- For the Adafruit Datalogger shield rev B open up **Examples->RTClib->pcf8523**
- For the older Adafruit Dataloggers, use **Examples->RTClib->ds1307**

Upload it to your Arduino with the datalogger shield on!



If you're having problems make sure you are running the right example! PCF8523 and DS1307 RTC chips are not identical so they have separate examples!

Now open up the Serial Console and make sure the baud rate is set correctly at **57600 baud** you should see the following:



```
RTC is NOT running!
2000/5/2 (Tuesday) 0:10:
 since midnight 1/1/1970 = 957226277s = 11079d
 now + 7d + 30s: 2000/5/9 12:41:23

2000/5/2 (Tuesday) 0:10:0
 since midnight 1/1/1970 = 957226200s = 11079d
 now + 7d + 30s: 2000/5/9 12:40:6

2000/5/2 (Tuesday) 0:10:3
 since midnight 1/1/1970 = 957226203s = 11079d
 now + 7d + 30s: 2000/5/9 12:40:9

2000/5/2 (Tuesday) 0:10:6
 since midnight 1/1/1970 = 957226206s = 11079d
 now + 7d + 30s: 2000/5/9 12:40:12

2000/5/2 (Tuesday) 0:10:9
 since midnight 1/1/1970 = 957226209s = 11079d
 now + 7d + 30s: 2000/5/9 12:40:15

2000/5/2 (Tuesday) 0:10:12
```

Whenever the RTC chip loses all power (including the backup battery) it will reset to an earlier date and report the time as 0:0:0 or similar. The DS1307 won't even count seconds (it's stopped).Whenever you set the time, this will kickstart the clock ticking.

So, basically, the upshot here is that you should never ever remove the battery once you've set the time. You shouldn't have to and the battery holder is very snug so unless the board is crushed, the battery won't 'fall out'

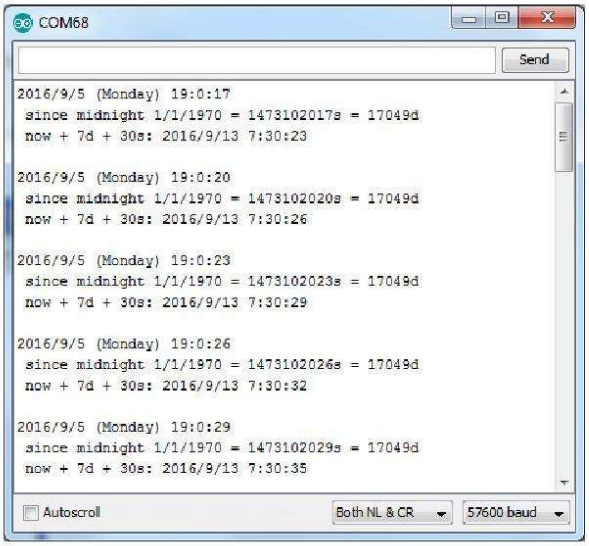
Setting the time

With the same sketch loaded, uncomment the line that starts with **RTC.adjust** like so:

```
  if (! rtc.initialized()) {
    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 is very cute, what it does is take the Date and Time according the computer you're using (right when you compile the code) and uses that to program the RTC. If your computer time is not set right you should fix that first. Then you must press the **Upload** button to compile and then immediately upload. If you compile and then upload later, the clock will be off by that amount of time.

Then open up the Serial monitor window to show that the time has been set



```
2016/9/5 (Monday) 19:0:17
 since midnight 1/1/1970 = 1473102017s = 17049d
 now + 7d + 30s: 2016/9/13 7:30:23

2016/9/5 (Monday) 19:0:20
 since midnight 1/1/1970 = 1473102020s = 17049d
 now + 7d + 30s: 2016/9/13 7:30:26

2016/9/5 (Monday) 19:0:23
 since midnight 1/1/1970 = 1473102023s = 17049d
 now + 7d + 30s: 2016/9/13 7:30:29

2016/9/5 (Monday) 19:0:26
 since midnight 1/1/1970 = 1473102026s = 17049d
 now + 7d + 30s: 2016/9/13 7:30:32

2016/9/5 (Monday) 19:0:29
 since midnight 1/1/1970 = 1473102029s = 17049d
 now + 7d + 30s: 2016/9/13 7:30:35
```

From now on, you won't have to ever set the time again: the battery will last 5 or more years

Reading the time

Now that the RTC is merrily ticking away, we'll want to query it for the time. Let's look at the sketch again to see how this is done

```
void loop () {
    DateTime now = rtc.now();

    Serial.print(now.year(), DEC);
    Serial.print('/');
    Serial.print(now.month(), DEC);
    Serial.print('/');
    Serial.print(now.day(), DEC);
    Serial.print(" (");
    Serial.print(daysOfTheWeek[now.dayOfTheWeek()]);
    Serial.print(") ");
    Serial.print(now.hour(), DEC);
    Serial.print(':');
    Serial.print(now.minute(), DEC);
    Serial.print(':');
    Serial.print(now.second(), DEC);
    Serial.println();
```

There's pretty much only one way to get the time using the RTClib, which is to call **now()**, a function that returns a DateTime object that describes the year, month, day, hour, minute and second when you called **now()**.

There are some RTC libraries that instead have you call something like **RTC.year()** and **RTC.hour()** to get the current year and hour. However, there's one problem where if you happen to ask for the minute right at **3:14:59** just before the next minute rolls over, and then the second right after the minute rolls over (so at **3:15:00**) you'll see the time as **3:14:00** which is a minute off. If you did it the other way around you could get **3:15:59** - so one minute off in the other direction.

Because this is not an especially unlikely occurance - particularly if you're querying the time pretty often - we take a 'snapshot' of the time from the RTC all at once and then we can pull it apart into **day()** or **second()** as seen above. It's a tiny bit more effort but we think its worth it to avoid mistakes!

We can also get a 'timestamp' out of the DateTime object by calling **unixtime** which counts the number of seconds (not counting leapseconds) since midnight, January 1st 1970

```
    Serial.print(" since 2000 = ");
    Serial.print(now.unixtime());
    Serial.print("s = ");
    Serial.print(now.unixtime() / 86400L);
    Serial.println("d");
```

Since there are 60*60*24 = 86400 seconds in a day, we can easily count days since then as well. This might be useful when you want to keep track of how much time has passed since the last query, making some math a lot easier (like checking if it's been 5 minutes later, just see if **unixtime()** has increased by 300, you dont have to worry about hour changes)

Using the SD Card

The other half of the data logger shield is the SD card. The SD card is how we store long term data. While the Arduino chip has a permanent EEPROM storage, its only a couple hundred bytes - tiny compared to a 2 gig SD card. SD cards are so cheap and easy to get, its an obvious choice for long term storage so we use them for the shield.

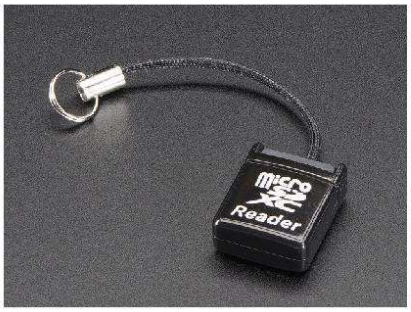
The shield kit doesn't come with an SD card but we carry one in the shop that is guaranteed to work (https://adafru.it/alH). Pretty much any SD card should work but be aware that some cheap cards are 'fakes' and can cause headaches.

4GB Blank SD/MicroSD Memory Card

$7.95
OUT OF STOCK

OUT OF STOCK

You'll also need a way to read and write from the SD card. Sometimes you can use your camera and MP3 player - when its plugged in you will be able to see it as a disk. Or you may need an SD card reader (http://adafru.it/939). The shield **doesnt** have the ability to display the SD card as a 'hard disk' like some MP3 players or games, the Arduino does not have the hardware for that, so you will need an external reader!



USB MicroSD Card Reader/Writer - microSD / microSDHC / microSDXC

$5.95
IN STOCK

ADD TO CART

Formatting under Windows/Mac

If you bought an SD card, chances are it's already pre-formatted with a FAT filesystem. However you may have problems with how the factory formats the card, or if it's an old card it needs to be reformatted. The Arduino SD library we use supports both **FAT16** and **FAT32** filesystems. If you have a very small SD card, say 8-32 Megabytes you might find it is formatted **FAT12** which isnt supported. You'll have to reformat these card. Either way, its **always** good idea to format the card before using, even if its new! Note that formatting will erase the card so save anything you want first

We strongly recommend you use the official SD card formatter utility - written by the SD association it solves many problems that come with bad formatting!

The official SD formatter is available from https://www.sdcard.org/downloads/formatter_4/ (https://adafru.it/cfL)

Download it and run it on your computer, there's also a manual linked from that page for use

https://adafru.it/cfL

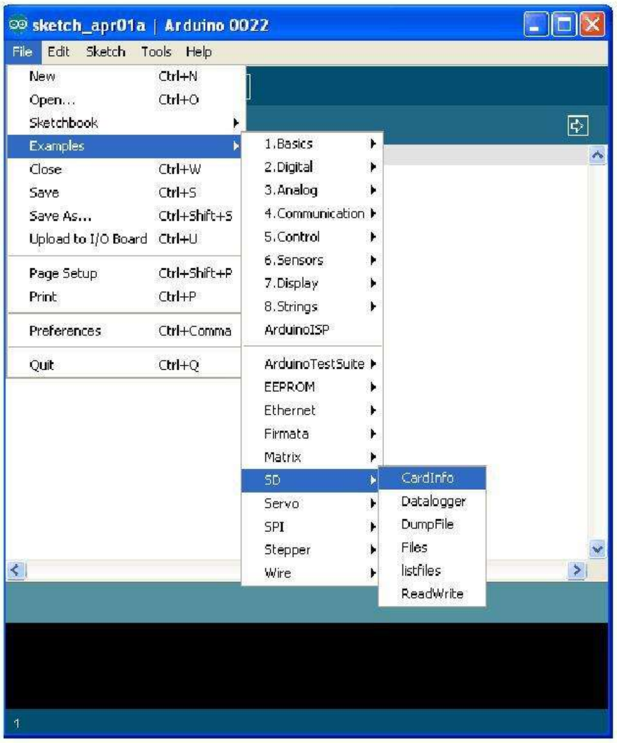
https://adafru.it/cfL

Get Card Info

The Arduino SD Card library has a built in example that will help you test the shield and your connections

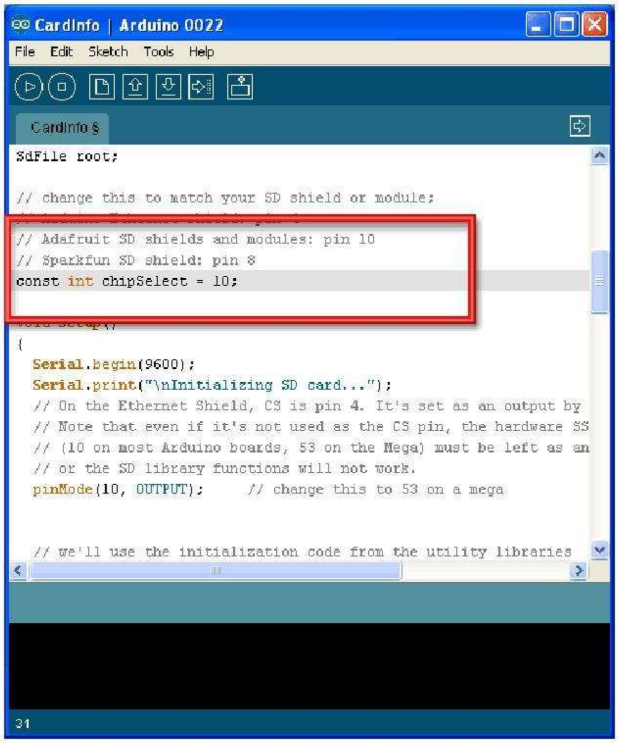
> If you have an older Datalogging shield without the SPI header connection **and** you are using a **Leonardo, Mega** or anything other than an UNO, you'll need to install a special version of the SD library (https://adafru.it/ref)

Open the file **CardInfo** example sketch in the **SD** library:



This sketch will not write any data to the card, just tell you if it managed to recognize it, and some information about it. This can be **very** useful when trying to figure out whether an SD card is supported. Before trying out a new card, please try out this sketch!

Go to the beginning of the sketch and make sure that the **chipSelect** line is correct, for the datalogger shield we 're using digital pin 10 so change it to 10!

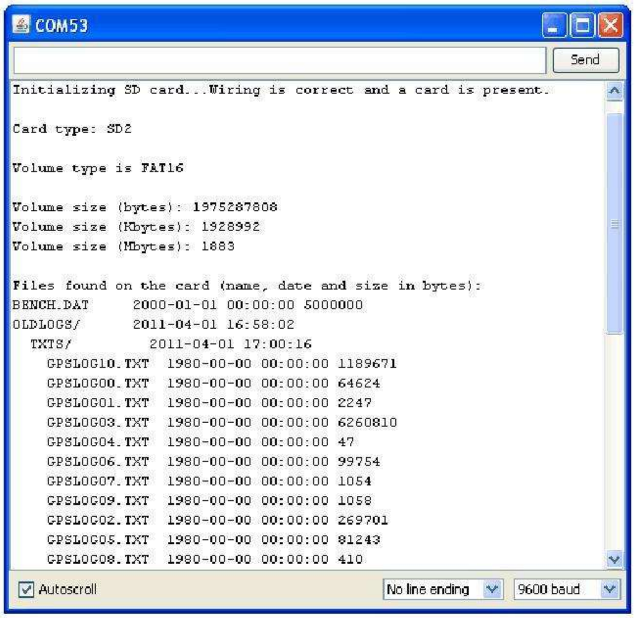


```
SdFile root;

// change this to match your SD shield or module;
// Adafruit SD shields and modules: pin 10
// Sparkfun SD shield: pin 8
const int chipSelect = 10;

{
  Serial.begin(9600);
  Serial.print("\nInitializing SD card...");
  // On the Ethernet Shield, CS is pin 4. It's set as an output by
  // Note that even if it's not used as the CS pin, the hardware SS
  // (10 on most Arduino boards, 53 on the Mega) must be left as an
  // or the SD library functions will not work.
  pinMode(10, OUTPUT);     // change this to 53 on a mega


  // we'll use the initialization code from the utility libraries
```

If you have the pre-rev B version of the Datalogger Shield, and you are using a Mega or Leonardo check here for now to adjust the pin setup (https://adafru.it/ref)

OK, now insert the SD card into the Arduino and upload the sketch

Open up the Serial Monitor and type in a character into the text box (& hit send) when prompted. You'll probably get something like the following:

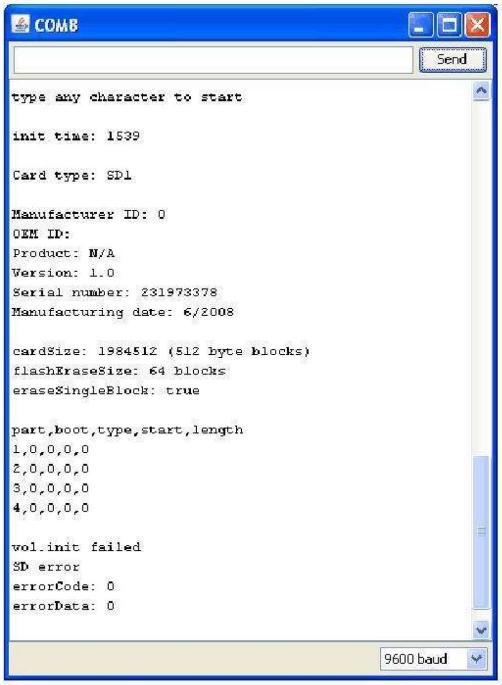


```
Initializing SD card...Wiring is correct and a card is present.

Card type: SD2

Volume type is FAT16

Volume size (bytes): 1975287808
Volume size (Kbytes): 1928992
Volume size (Mbytes): 1883

Files found on the card (name, date and size in bytes):
BENCH.DAT     2000-01-01 00:00:00 5000000
OLDLOGS/      2011-04-01 16:58:02
  TXTS/         2011-04-01 17:00:16
    GPSLOG10.TXT  1980-00-00 00:00:00 1189671
    GPSLOG00.TXT  1980-00-00 00:00:00 64624
    GPSLOG01.TXT  1980-00-00 00:00:00 2247
    GPSLOG03.TXT  1980-00-00 00:00:00 6260810
    GPSLOG04.TXT  1980-00-00 00:00:00 47
    GPSLOG06.TXT  1980-00-00 00:00:00 99754
    GPSLOG07.TXT  1980-00-00 00:00:00 1054
    GPSLOG09.TXT  1980-00-00 00:00:00 1058
    GPSLOG02.TXT  1980-00-00 00:00:00 269701
    GPSLOG05.TXT  1980-00-00 00:00:00 81243
    GPSLOG08.TXT  1980-00-00 00:00:00 410
```

☑ Autoscroll — No line ending — 9600 baud

Its mostly gibberish, but its useful to see the **Volume type is FAT16** part as well as the size of the card (about 2 GB which is what it should be) etc.

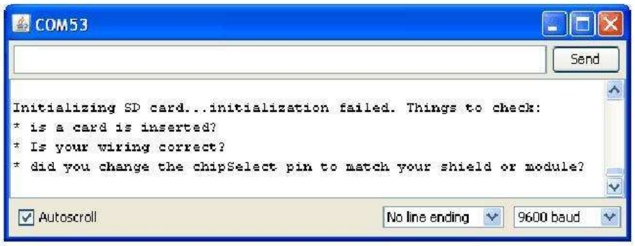
If you have a bad card, which seems to happen more with ripoff version of good brands, you might see:



```
type any character to start

init time: 1539

Card type: SD1

Manufacturer ID: 0
OEM ID:
Product: N/A
Version: 1.0
Serial number: 231973378
Manufacturing date: 6/2008

cardSize: 1984512 (512 byte blocks)
flashEraseSize: 64 blocks
eraseSingleBlock: true

part,boot,type,start,length
1,0,0,0,0
2,0,0,0,0
3,0,0,0,0
4,0,0,0,0

vol.init failed
SD error
errorCode: 0
errorData: 0
```

The card mostly responded, but the data is all bad. Note that the **Product ID** is **"N/A"** and there is no **Manufacturer ID** or **OEM ID**. This card returned some SD errors. Its basically a bad scene, I only keep this card around to use as an example of a bad card! If you get something like this (where there is a response but its corrupted) you should toss the card

Finally, try taking out the SD card and running the sketch again, you'll get the following,



```
Initializing SD card...initialization failed. Things to check:
* is a card is inserted?
* Is your wiring correct?
* did you change the chipSelect pin to match your shield or module?
```

It couldn't even initialize the SD card. This can also happen if there's a soldering error or if the card is *really* damaged

If you're having SD card problems, we suggest using the SD formatter mentioned above first to make sure the card is clean and ready to use!

Light and Temperature Logger

Introduction

OK now that we have introduced both the RTC and the SD card and verified that they're working, we can move onto logging!

We'll use a pretty good & detailed demonstration to show off the capabilities of this most awesome data logging shield: We'll log both temperature and relative light levels to determine:

1. How much does the temperature in a fridge vary as the compressor turns on and off?
2. Does keeping the door open cause a big temperature drop? How long does it take for it to cool down?
3. Does the light inside *really* turn off when the door is closed?

Build It!

Items you'll need:

- Arduino (of course!) a Atmega328 type is best (http://adafru.it/50)- we always recommend going with an official 'classic' Arduino such as the Uno.
- Adafruit data logger shield (http://adafru.it/1141) - assembled
- SD card formatted for FAT (http://adafru.it/102) and tested using our example sketch (https://adafru.it/clN)
- CdS photocell (http://adafru.it/161) and a matching 10K pulldown resistor
- Temperature sensor with analog out, such as TMP36 (http://adafru.it/165)
- Battery pack such as a 6-AA 'brick' and a 2.1mm DC jack. (http://adafru.it/248)
- or you can use a 9V clip for a power supply(http://adafru.it/80) but a 9V powered logger will last only a couple hours so we suggest 6xAA's
- Some 22 AWG wire (https://adafru.it/c79), soldering iron, solder (https://adafru.it/doU), etc.

You can get most everything in that list in a discounted pack in the Adafruit shop!(http://adafru.it/249)

adafruit industries
Data Logger Shield Kit v1.0
Data logging with timestamps for your Arduino
To build this kit and download files, visit:
http://www.adafruit.com/logshield
2GB
2 GB GO

The sensors

We'll use two basic sensors to log data, a CdS photocell to track light (http://adafru.it/161) (this will tell us when the door has been opened) and a semiconductor temperature sensor to log the ambient fridge temperature. (http://adafru.it/165)

We have two great tutorials for these sensors on our site, if you haven't used them before or need some refreshment, please read them now!

https://adafru.it/reg

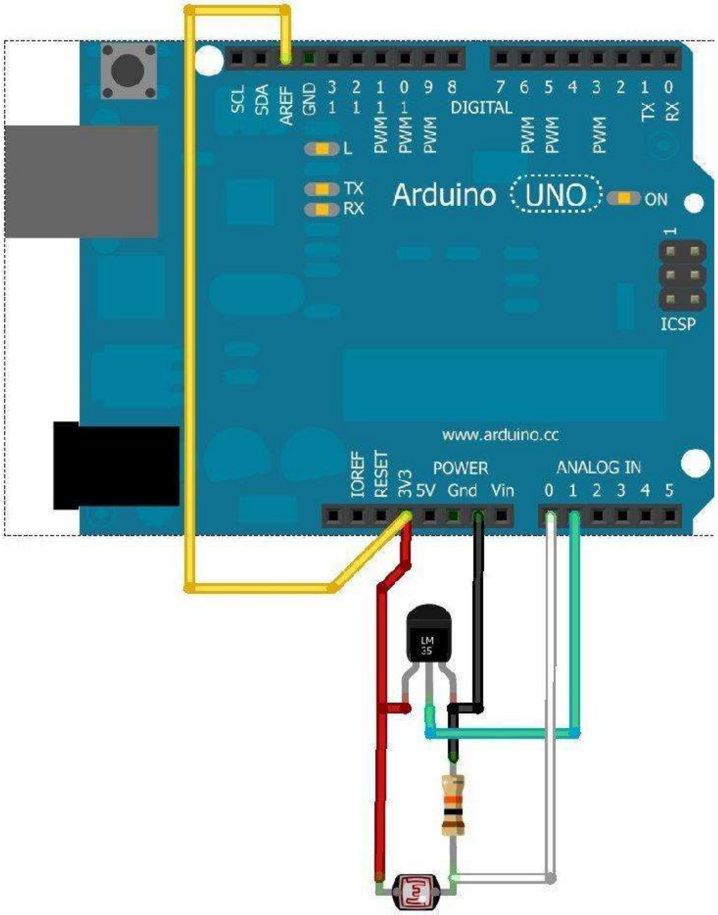
https://adafru.it/reg

https://adafru.it/reh

https://adafru.it/reh

We will wire the sensors as shown in the diagram below.

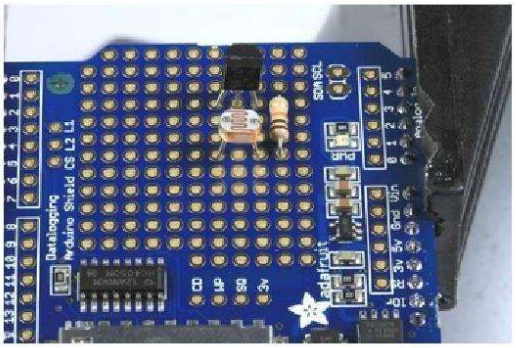
Note that we connect **ARef**, the power pin of the temp sensor, and the light sensor to **3.3V** *not to 5.0V* - we do this because the 5V line is very noisy and the 3.3V regulator is better filtered. In the actual board we used the 3.3V line from the datalogger's regulator, see the images below - in theory its the same as the one off of the Arduino but we trust ours more.



Made with Fritzing.org

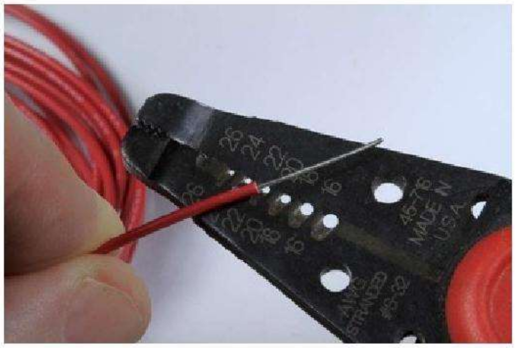
Wiring it up

The prototyping area on the board is a simple array of holes with soldering pads. The steps below show how we built this circuit and illustrate some some basic circuit prototyping techniques. For clarity, we will use the same color wire as shown in the circuit diagram above:



Position the sensors

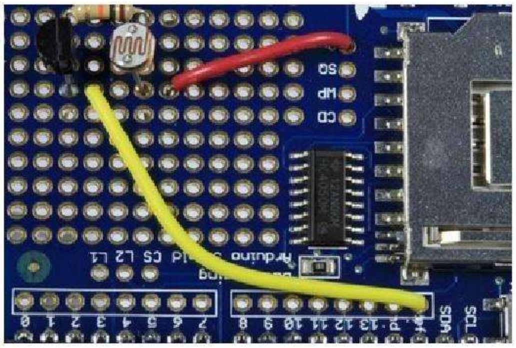
The sensors could go anywhere on the prototyping area, but we chose this arrangement to simplify connections between the components later on.



Prepare some jumpers

Measure a piece of wire (red) long enough to reach from the 3v breakout hole to 1/2" past the temperature sensor. Strip about 3/4" from one end, and about 1/4" from the other.

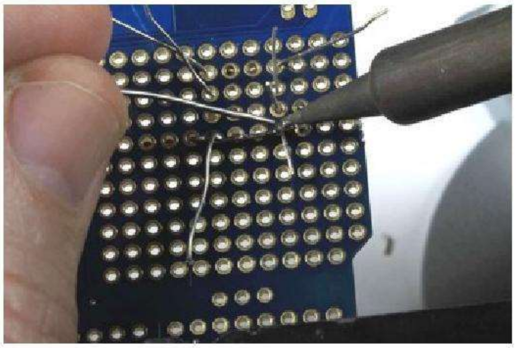
Measure another one (yellow) long enough to reach from the AREF pin to the hole between the two sensors. Strip 1/2" from one end and 1/4" from the other.



Install the Jumpers

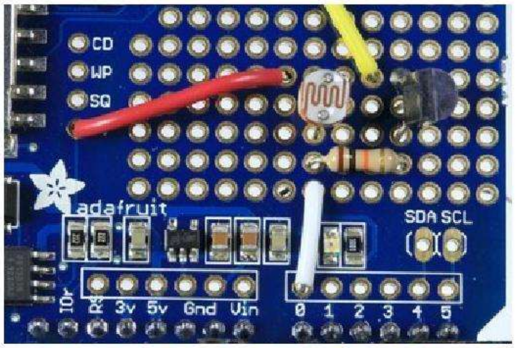
Place the jumpers as shown, with the long stripped ends nearest the sensors.

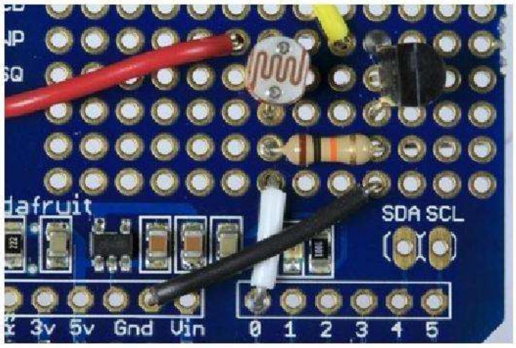
Since there are no signal traces between the holes in the prototyping area, we will use the long stripped ends to join the legs of the components on the board.

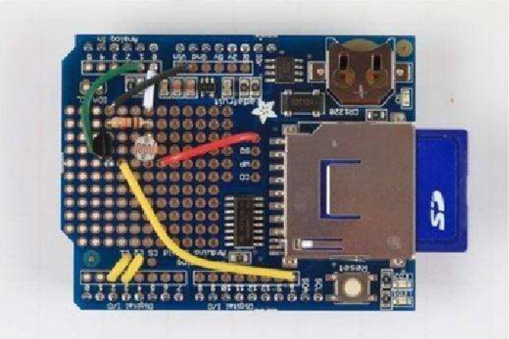


Make the connections

- Solder the first jumper (red) to the 3v hole.
- Bend the stripped end of the wire so it rests next to the legs of the light sensor, the temperature sensor and the end of the AREF jumper.
- Fold the sensor legs and AREF jumper legs over the 3v jumper and solder to make the connection.





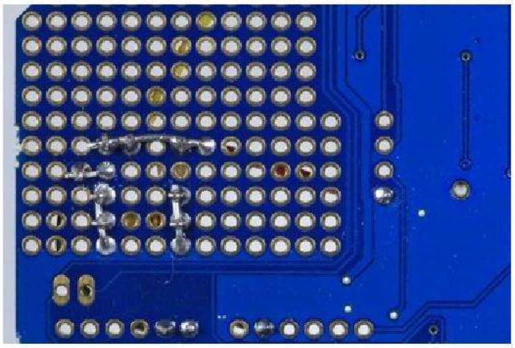


Add more jumpers for the Sensors

- From Analog Pin 0 to the hole near the light sensor and resistor. (white)
- From GND to the hole next to the other end of the resistor (black)
- From the Analog pin 1 to the hole next to the center pin of the temperature sensor (green)

And also for the LEDs

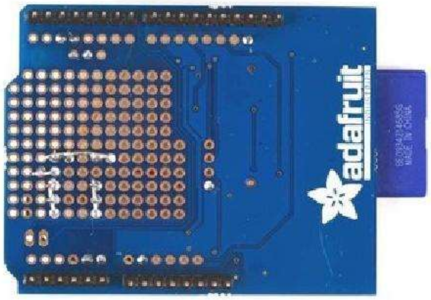
- From L1 to Digital Pin 2 (yellow)
- From L2 to Digital Pin 3 (yellow)

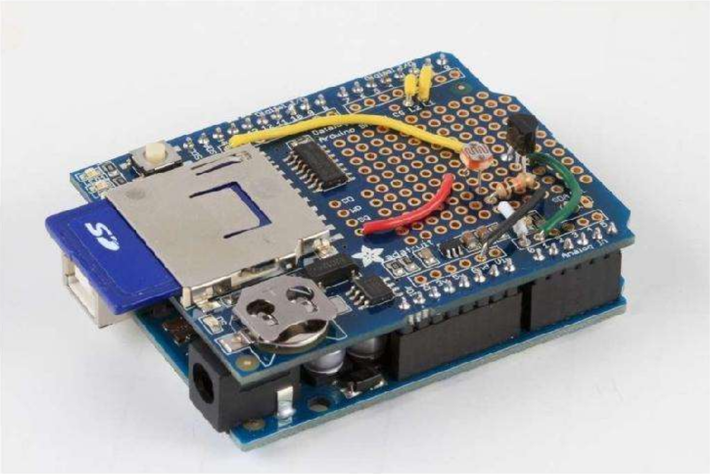


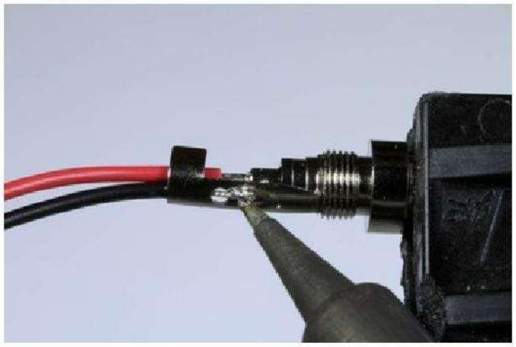
Solder and trim all connections

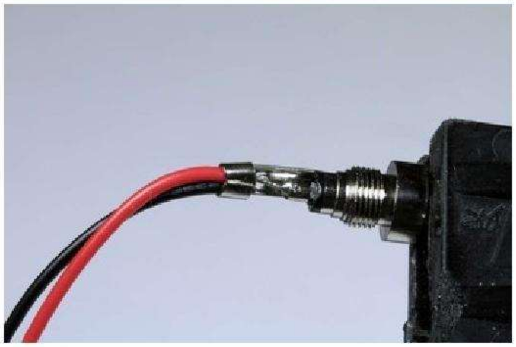
Using the same technique of folding the component legs over the jumper - make all connections as shown in the wiring diagram.

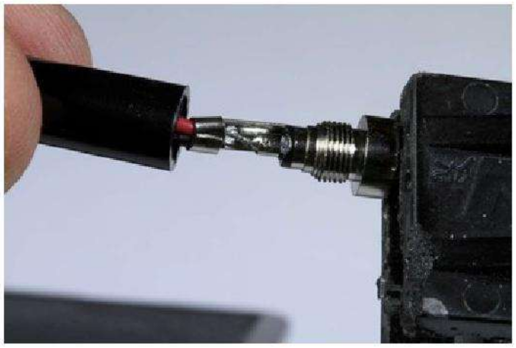
Make sure that all connections are soldered. Also solder wires and component legs to the board where they pass through the holes.











Prepare the Battery Pack

- Place the black plastic ferrule from the connector over the battery pack wires.
- Solder the red wire from the battery pack to the center pin
- Solder the the black wire to the outer barrel.
- Crimp to hold the wires securely
- Screw the black plastic ferrule on to cover the solder joints.

Now your Light Temp Logger is wired and ready for testing!

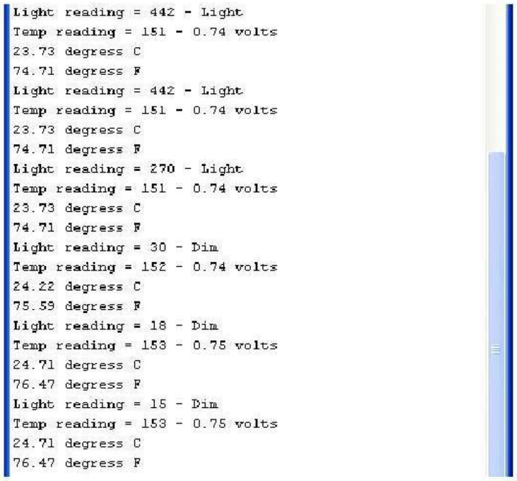
Use It!

Sensor test

We'll now test the sensors, using this sketch which is a bit of a mashup of the two examples in our tutorials (https://adafru.it/c7d)

```
#include <SPI.h>
#include <SD.h>

/* Sensor test sketch
  for more information see http://www.ladyada.net/make/logshield/lighttemp.html
  */

#define aref_voltage 3.3         // we tie 3.3V to ARef and measure it with a multimeter!

int photocellPin = 0;     // the cell and 10K pulldown are connected to a0
int photocellReading;     // the analog reading from the analog resistor divider

//TMP36 Pin Variables
int tempPin = 1;        //the analog pin the TMP36's Vout (sense) pin is connected to
                        //the resolution is 10 mV / degree centigrade with a
                        //500 mV offset to allow for negative temperatures
int tempReading;        // the analog reading from the sensor

void setup(void) {
  // We'll send debugging information via the Serial monitor
  Serial.begin(9600);

  // If you want to set the aref to something other than 5v
  analogReference(EXTERNAL);
}


void loop(void) {
  photocellReading = analogRead(photocellPin);

  Serial.print("Light reading = ");
  Serial.print(photocellReading);     // the raw analog reading

  // We'll have a few threshholds, qualitatively determined
  if (photocellReading < 10) {
    Serial.println(" - Dark");
  } else if (photocellReading < 200) {
    Serial.println(" - Dim");
  } else if (photocellReading < 500) {
    Serial.println(" - Light");
  } else if (photocellReading < 800) {
    Serial.println(" - Bright");
  } else {
    Serial.println(" - Very bright");
  }

  tempReading = analogRead(tempPin);

  Serial.print("Temp reading = ");
  Serial.print(tempReading);     // the raw analog reading
  // converting that reading to voltage, which is based off the reference voltage
  float voltage = tempReading * aref_voltage / 1024;

  // print out the voltage
  Serial.print(" - ");
  Serial.print(voltage); Serial.println(" volts");

  // now print out the temperature
  float temperatureC = (voltage - 0.5) * 100 ;  //converting from 10 mv per degree wit 500 mV offset
                                               //to degrees ((volatge - 500mV) times 100)
  Serial.print(temperatureC); Serial.println(" degrees C");

  // now convert to Fahrenheight
  float temperatureF = (temperatureC * 9 / 5) + 32;
  Serial.print(temperatureF); Serial.println(" degrees F");

  delay(1000);
}
```

OK upload this sketch and check the Serial monitor again



```
Light reading = 442 - Light
Temp reading = 151 - 0.74 volts
23.73 degress C
74.71 degress F
Light reading = 442 - Light
Temp reading = 151 - 0.74 volts
23.73 degress C
74.71 degress F
Light reading = 270 - Light
Temp reading = 151 - 0.74 volts
23.73 degress C
74.71 degress F
Light reading = 30 - Dim
Temp reading = 152 - 0.74 volts
24.22 degress C
75.59 degress F
Light reading = 18 - Dim
Temp reading = 153 - 0.75 volts
24.71 degress C
76.47 degress F
Light reading = 15 - Dim
Temp reading = 153 - 0.75 volts
24.71 degress C
76.47 degress F
```

In my workroom, I got about 24 degrees C and a 'light measurement' of about 400 - remember that while the temperature sensor gives an 'absolute' reading in C or F, the light sensor is not precise and can only really give rough readings.

Once you've verified that the sensors are wired up correctly & running its time to get to the logging!

Logging sketch

Download the light and temperature logging sketch from GitHub (https://adafru.it/c7e). Insert the SD card.

Look at the top of the sketch for this section and uncomment whichever line is relevant. Check the RTC page for details if you're not sure which one you have. (https://adafru.it/rei)

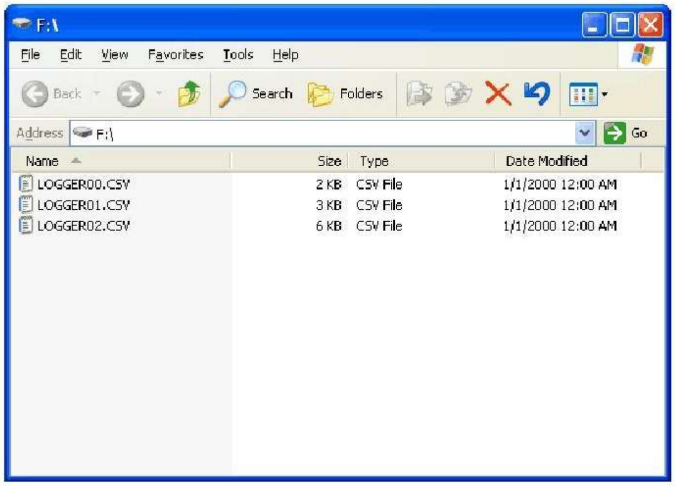
```
/************** if you have a DS1307 uncomment this line **************/

//RTC_DS1307 RTC; // define the Real Time Clock object

/************** if you have a PCF8523 uncomment this line **************/

//RTC_PCF8523 RTC; // define the Real Time Clock object

/***********************************************************************/
```

Upload the sketch to your Arduino. We'll now test it out while still 'tethered' to the computer

While the Arduno is still connected, blinking and powered, place your hand over the photocell for a few seconds, then shine a flashlight on it. You should also squeeze the temp sensor with your fingers to heat it up

Plotting with a spreadsheet

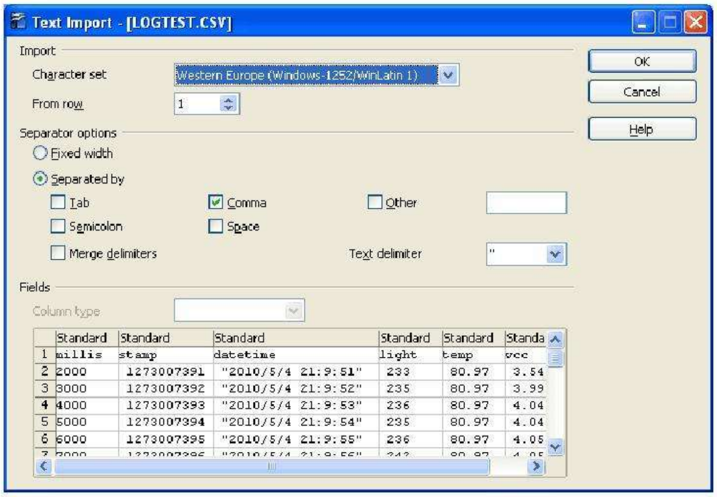
When you're ready to check out the data, unplug the Arduino and put the SD card into your computer's card reader. You'll see a at least one and perhaps a couple files, one for each time the logger ended up running



Name	Size	Type	Date Modified
LOGGER00.CSV	2 KB	CSV File	1/1/2000 12:00 AM
LOGGER01.CSV	3 KB	CSV File	1/1/2000 12:00 AM
LOGGER02.CSV	6 KB	CSV File	1/1/2000 12:00 AM

We'll open the most recent one. If you want to use the same logfile used in the graphing demos, click here to download it (https://adafru.it/cny).

The quickest way to look at the data is using something like OpenOffice or Excel, where you can open the .csv file and have it imported directly into the spreadsheet



Import

Character set: Western Europe (Windows-1252/WinLatin 1)

From row: 1

Separator options

○ Fixed width

● Separated by

☐ Tab ☑ Comma ☐ Other

☐ Semicolon ☐ Space

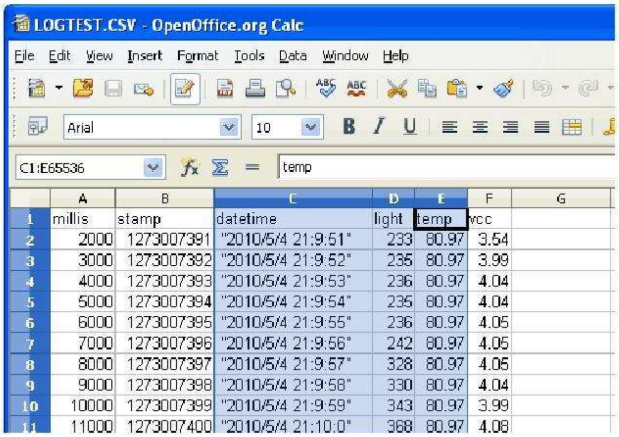
☐ Merge delimiters Text delimiter: "

Fields

Column type

	Standard	Standard	Standard	Standard	Standard	Standa
1	millis	stamp	datetime	light	temp	vcc
2	2000	1273007391	"2010/5/4 21:9:51"	233	80.97	3.54
3	3000	1273007392	"2010/5/4 21:9:52"	235	80.97	3.99
4	4000	1273007393	"2010/5/4 21:9:53"	236	80.97	4.04
5	5000	1273007394	"2010/5/4 21:9:54"	235	80.97	4.04
6	6000	1273007395	"2010/5/4 21:9:55"	236	80.97	4.05

You can then perform some graphing by selecting the columns of data

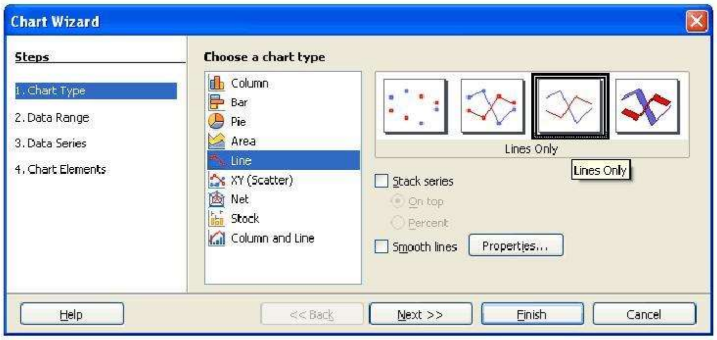


LOGTEST.CSV - OpenOffice.org Calc
File Edit View Insert Format Tools Data Window Help
Arial
10

C1:E65536 = temp

	A	B	C	D	E	F	G
1	millis	stamp	datetime	light	temp	vcc	
2	2000	1273007391	"2010/5/4 21:9:51"	233	80.97	3.54	
3	3000	1273007392	"2010/5/4 21:9:52"	235	80.97	3.99	
4	4000	1273007393	"2010/5/4 21:9:53"	236	80.97	4.04	
5	5000	1273007394	"2010/5/4 21:9:54"	235	80.97	4.04	
6	6000	1273007395	"2010/5/4 21:9:55"	236	80.97	4.05	
7	7000	1273007396	"2010/5/4 21:9:56"	242	80.97	4.05	
8	8000	1273007397	"2010/5/4 21:9:57"	328	80.97	4.05	
9	9000	1273007398	"2010/5/4 21:9:58"	330	80.97	4.04	
10	10000	1273007399	"2010/5/4 21:9:59"	343	80.97	3.99	
11	11000	1273007400	"2010/5/4 21:10:0"	368	80.97	4.08	

Clicking the **Chart** button and using Lines (we think they are the best for such graphs)



Choose a chart type

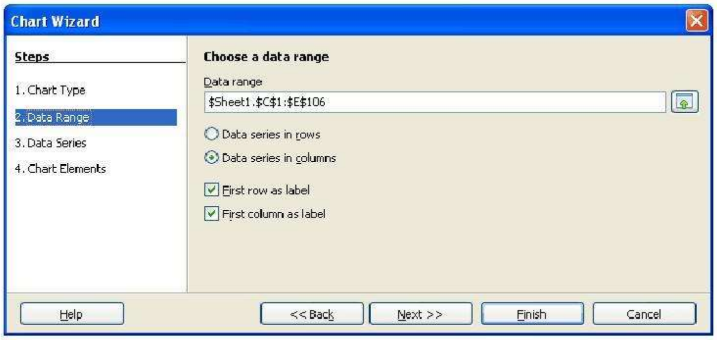
☐ Stack series

○ On top

○ Percent

☐ Smooth lines

Setting the **First Column as label**



Choose a data range

Data range

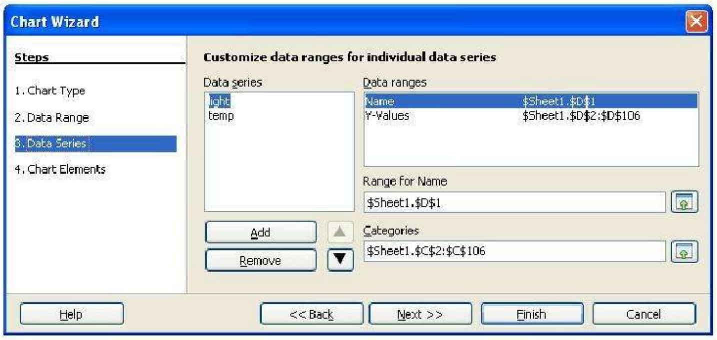
$Sheet1.$C$1:$E$106

○ Data series in rows

● Data series in columns

☑ First row as label

☑ First column as label



Customize data ranges for individual data series

Data series

light

temp

Data ranges	
Name	$Sheet1.$D$1
Y-Values	$Sheet1.$D$2:$D$106

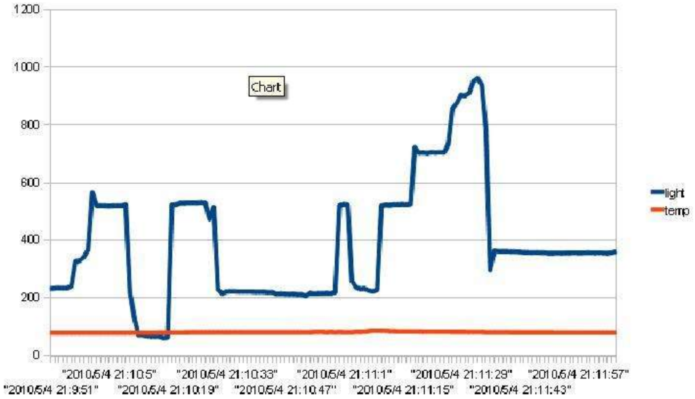
Range for Name

$Sheet1.$D$1

Categories

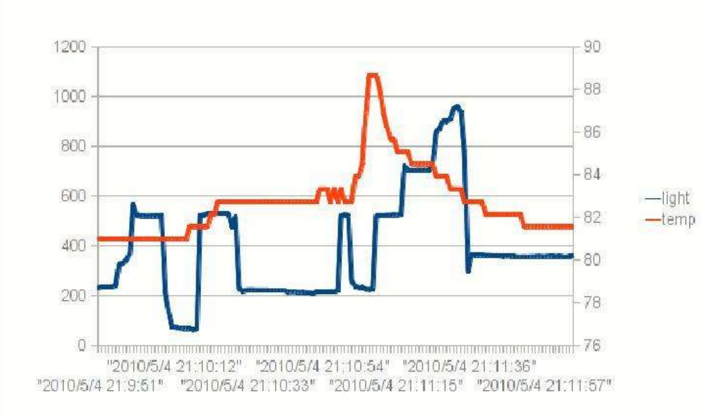
$Sheet1.$C$2:$C$106

Which will generate this graph

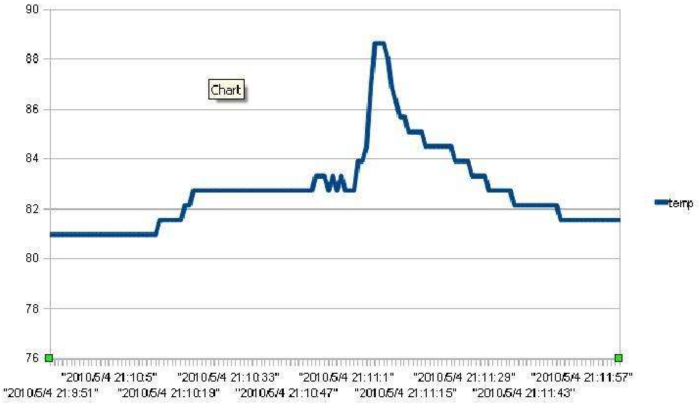


You can see pretty clearly how I shaded the sensor and then shone a flashlight on it.

You can make the graph display both with different axes (since the change in temperature is a different set of units. Select the temp line (red), right-click and choose **Format Data Series**. In the **Options** tab, **Align data series to Secondary Y-axis**.



Or you can make another graph with only the **temp** data



Now you can see clearly how I warmed up the sensor by holding it between my fingers

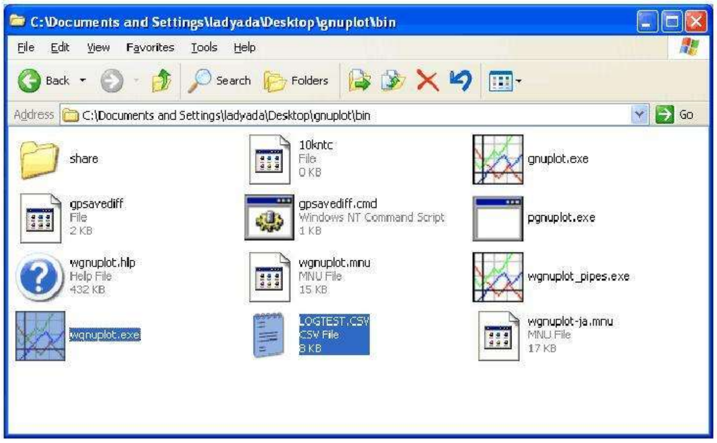
Using Gnuplot

Gnuplot is an free (but not open source?), ultra-powerful plotting program. Its also a real pain to use! But if you can't afford a professional math/plotting package such as Mathematica or Matlab, Gnuplot can do a lot!

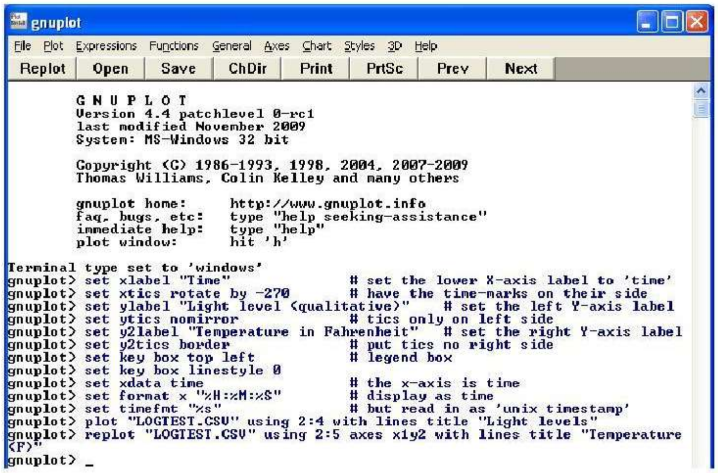
We're not good enough to provide a full tutorial on gnuplot, here are a few links we found handy. Google will definitely help you find even more tutorials and links. Mucking about is the best teacher, too!

- http://www.cs.hmc.edu/~vrable/gnuplot/using-gnuplot.html (https://adafru.it/c7i)
- http://www.duke.edu/~hpgavin/gnuplot.html (https://adafru.it/c7k)
- http://www.ibm.com/developerworks/library/l-gnuplot/ (https://adafru.it/c7m)

We found the following commands executed in order will generate a nice graph of this data, be sure to put LOGTEST.CSV in the same directory as **wgnuplot.exe** (or if you know how to reference directories, you can put it elsewhere)

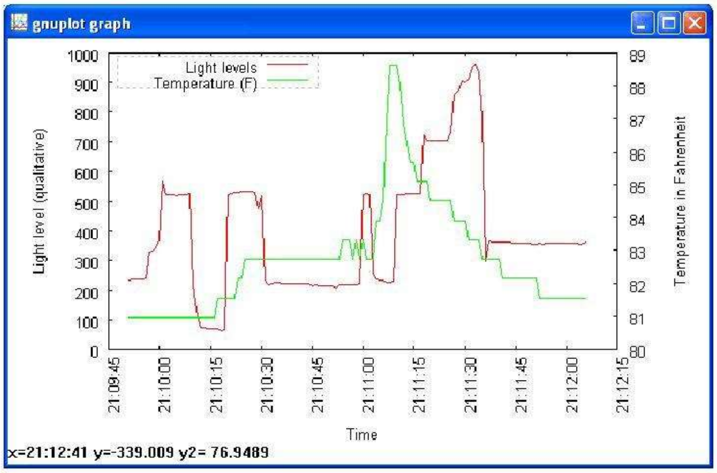

C:\Documents and Settings\ladyada\Desktop\gnuplot\bin
File Edit View Favorites Tools Help
Back
Search
Folders
Address C:\Documents and Settings\ladyada\Desktop\gnuplot\bin
Go
share
10kntc File 0 KB
gnuplot.exe
gpsavediff File 2 KB
gpsavediff.cmd Windows NT Command Script 1 KB
pgnuplot.exe
wgnuplot.hlp Help File 432 KB
wgnuplot.mnu MNU File 15 KB
wgnuplot_pipes.exe
wgnuplot.exe
LOGTEST.CSV CSV File 8 KB
wgnuplot-ja.mnu MNU File 17 KB

```
set xlabel "Time"              # set the lower X-axis label to 'time'

set xtics rotate by -270       # have the time-marks on their side


set ylabel "Light level (qualitative)"    # set the left Y-axis label

set ytics nomirror             # tics only on left side

set y2label "Temperature in Fahrenheit"   # set the right Y-axis label
set y2tics border              # put tics no right side

set key box top left           # legend box
set key box linestyle 0

set xdata time                 # the x-axis is time
set format x "%H:%M:%S"        # display as time
set timefmt "%s"               # but read in as 'unix timestamp'

plot "LOGTEST.CSV" using 2:4 with lines title "Light levels"
replot "LOGTEST.CSV" using 2:5 axes x1y2 with lines title "Temperature (F)"
```


gnuplot
File Plot Expressions Functions General Axes Chart Styles 3D Help
Replot Open Save ChDir Print PrtSc Prev Next

```
        G N U P L O T
        Version 4.4 patchlevel 0-rc1
        last modified November 2009
        System: MS-Windows 32 bit

        Copyright (C) 1986-1993, 1998, 2004, 2007-2009
        Thomas Williams, Colin Kelley and many others

        gnuplot home:     http://www.gnuplot.info
        faq, bugs, etc:   type "help seeking-assistance"
        immediate help:   type "help"
        plot window:      hit 'h'

Terminal type set to 'windows'
gnuplot> set xlabel "Time"              # set the lower X-axis label to 'time'
gnuplot> set xtics rotate by -270       # have the time-marks on their side
gnuplot> set ylabel "Light level (qualitative)"    # set the left Y-axis label
gnuplot> set ytics nomirror             # tics only on left side
gnuplot> set y2label "Temperature in Fahrenheit"   # set the right Y-axis label
gnuplot> set y2tics border              # put tics no right side
gnuplot> set key box top left           # legend box
gnuplot> set key box linestyle 0
gnuplot> set xdata time                 # the x-axis is time
gnuplot> set format x "%H:%M:%S"        # display as time
gnuplot> set timefmt "%s"               # but read in as 'unix timestamp'
gnuplot> plot "LOGTEST.CSV" using 2:4 with lines title "Light levels"
gnuplot> replot "LOGTEST.CSV" using 2:5 axes x1y2 with lines title "Temperature
(F)"
gnuplot> _
```

Which makes this:



x=21:12:41 y=-339.009 y2= 76.9489

Note the cool double-sided y-axis scales! You can zoom in on stuff pretty easily too.

Other plotters

Our friend John also suggests Live-Graph as a free plotting program(https://adafru.it/c7o) (https://adafru.it/c7o) - we haven't tried it but its worth looking at if you need to do a lot of plotting!

Portable logging

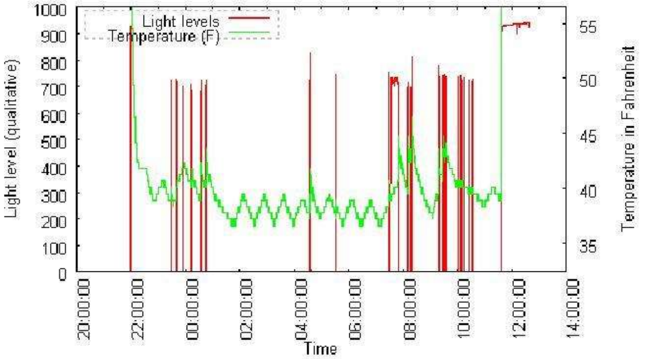
Of course, having a datalogger thats chained to a desktop computer isn't that handy. We can make a portable logger with the addition of a battery pack. The cheapest way to get a good amount of power is to use 6 AA batteries. I made one here with rechargables and a 6xAA battery holder (http://adafru.it/248). It ran the Arduino logging once a second for 18.5 hours. If you use alkalines you could easily get 24 hours or more.

Fridge logging

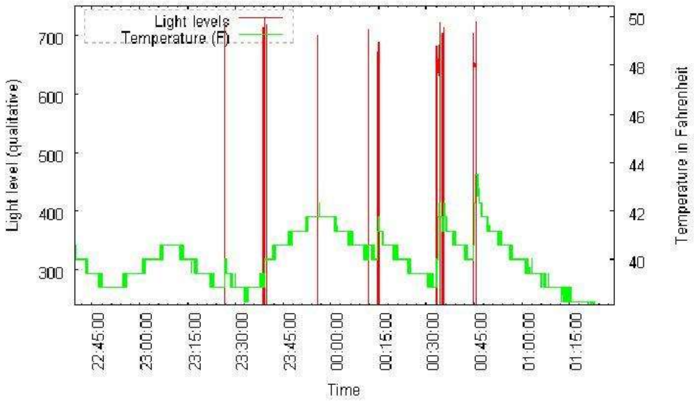
With my portable logger ready, its time to do some Fridge Loggin'! Both were placed in the fridge, in the center of the middle shelf.

I placed it in around 10PM and then removed it around noon the next day. If you don't have a fridge handy, you can grab the data from this zip file and use that (https://adafru.it/cnz).

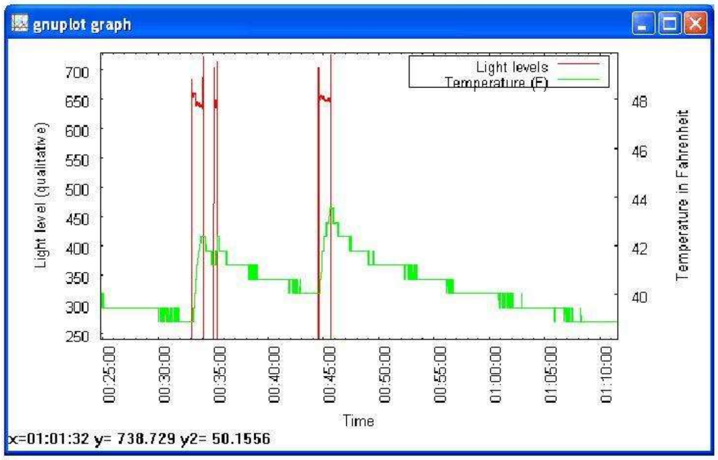
Here is the logged data:



You can see in the middle and end the temp and light levels are very high because the logger was outside the fridge. The green line is the temperature so you can see the temperature slowly rising and then the compressor kicking in every half hour or so. The red lines indicate when the door was opened. This night was a more insominac one than normal!



Zooming into the plot at about 12:40AM, we can see how the temperature climbs whenever the door is open, even in a few seconds it can climb 4 degrees very quickly!



x=01:01:32 y= 738.729 y2= 50.1556

Conclusion!

OK that was a detailed project but its a good one to test your datalogging abilities, especially since its harder to fix bugs in the field. In general, we suggest trying other sensors and testing them at home if possible. Its also a good idea to log more data than you need, and use a software program to filter anything you dont need. For example, we dont use the VCC log but if you're having strange sensor behavior, it may give you clues if your battery life is affecting it.

Code Walkthrough

Introduction

This is a walkthrough of the Light and Temperature Logging sketch. Its long and detailed so we put it here for your perusal. We strongly suggest reading through it, the code is very versatile and our text descriptions should make it clear why everything is there!

Download the complete file here (https://adafru.it/c7e):

Includes and Defines

```
#include "SD.h"
#include <Wire.h>
#include "RTClib.h"
```

OK this is the top of the file, where we include the three libraries we'll use: the **SD** library to talk to the card, the **Wire** library that helps the Arduino with i2c and the **RTClib** for chatting with the real time clock

```
// A simple data logger for the Arduino analog pins
#define LOG_INTERVAL  1000 // mills between entries
#define ECHO_TO_SERIAL   1 // echo data to serial port
#define WAIT_TO_START    0 // Wait for serial input in setup()


// the digital pins that connect to the LEDs
#define redLEDpin 3
#define greenLEDpin 4

// The analog pins that connect to the sensors
#define photocellPin 0           // analog 0
#define tempPin 1                // analog 1
```

Next are all the "defines" - the constants and tweakables.

- `LOG_INTERVAL` is how many milliseconds between sensor readings. 1000 is 1 second which is not a bad starting point
- `ECHO_TO_SERIA`L determines whether to send the stuff thats being written to the card also out to the Serial monitor. This makes the logger a little more sluggish and you may want the serial monitor for other stuff. On the other hand, its hella useful. We'll set this to **1** to keep it on. Setting it to **0** will turn it off
- `WAIT_TO_START` means that you have to send a character to the Arduino's Serial port to kick start the logging. If you have this on you basically can't have it run away from the computer so we'll keep it off (set to **0**) for now. If you want to turn it on, set this to **1**

The other defines are easier to understand, as they are just pin defines

- `redLEDpin` is whatever you connected to the Red LED on the logger shield
- `greenLEDpin` is whatever you connected to the Green LED on the logger shield
- `photocellPin` is the analog input that the CdS cell is wired to
- `tempPin` is the analog input that the TMP36 is wired to

Objects and error()

```
RTC_DS1307 RTC; // define the Real Time Clock object


// for the data logging shield, we use digital pin 10 for the SD cs line
const int chipSelect = 10;

// the logging file
File logfile;

void error(char *str)
{
  Serial.print("error: ");
  Serial.println(str);

  // red LED indicates error
  digitalWrite(redLEDpin, HIGH);

  while(1);
}
```

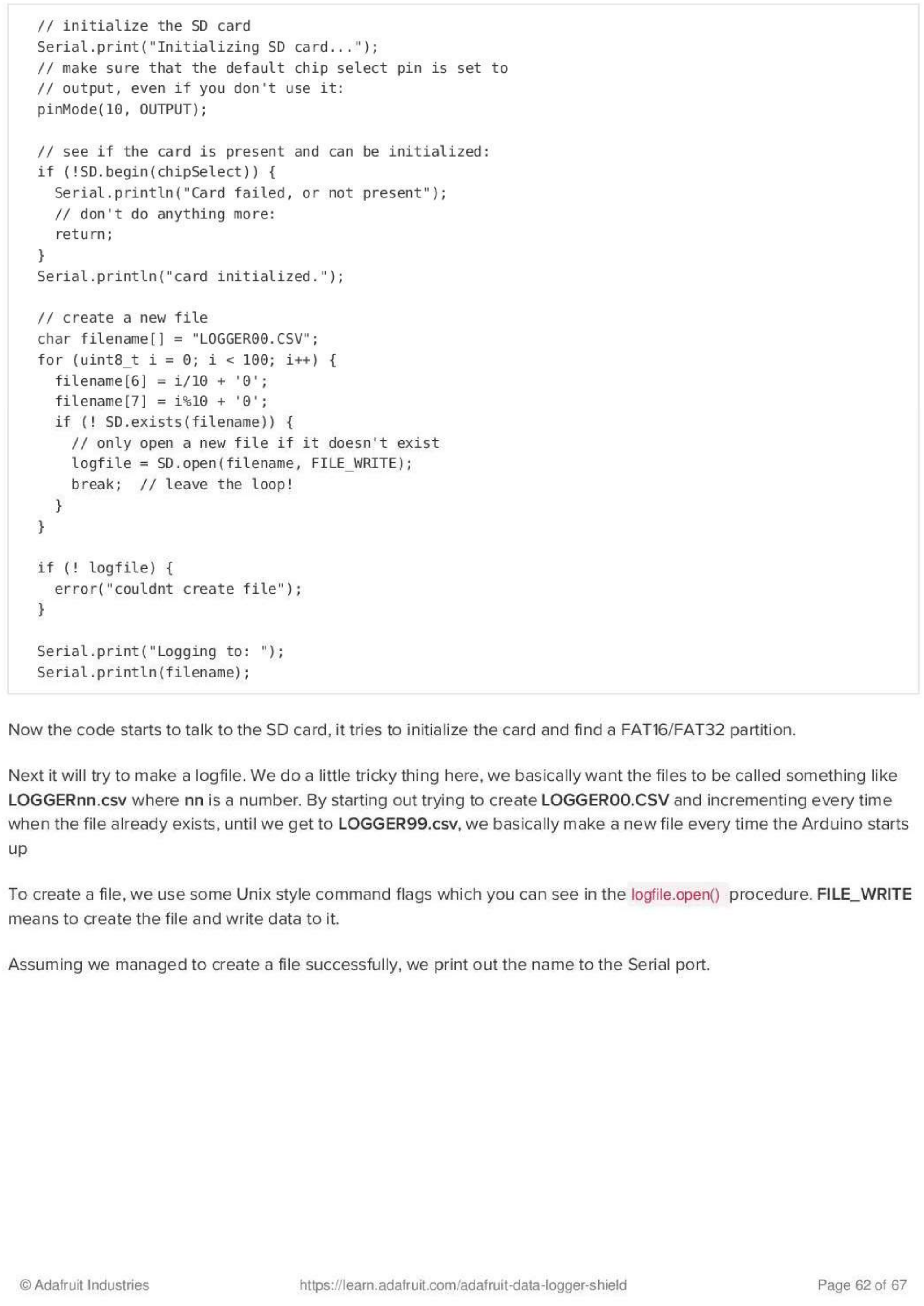
Next up we've got all the objects for the RTC, and the SD card chip select pin. For all our shields we use **pin 10** for SD card chip select lines

Next is the `error()` function, which is just a shortcut for us, we use it when something Really Bad happened, like we couldn't write to the SD card or open it. It prints out the error to the Serial Monitor, turns on the red error LED, and then sits in a `while(1);` loop forever, also known as a **halt**

Setup

```
void setup(void)
{
  Serial.begin(9600);
  Serial.println();

#if WAIT_TO_START
  Serial.println("Type any character to start");
  while (!Serial.available());
#endif //WAIT_TO_START
```

K now we are onto the code. We begin by initializing the Serial port at 9600 baud. If we set `WAIT_TO_START` to anything but **0**, the Arduino will wait until the user types something in. Otherwise it goes ahead to the next part



```
  // initialize the SD card
  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(10, OUTPUT);

  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
    // don't do anything more:
    return;
  }
  Serial.println("card initialized.");

  // create a new file
  char filename[] = "LOGGER00.CSV";
  for (uint8_t i = 0; i < 100; i++) {
    filename[6] = i/10 + '0';
    filename[7] = i%10 + '0';
    if (! SD.exists(filename)) {
      // only open a new file if it doesn't exist
      logfile = SD.open(filename, FILE_WRITE);
      break;  // leave the loop!
    }
  }

  if (! logfile) {
    error("couldnt create file");
  }

  Serial.print("Logging to: ");
  Serial.println(filename);
```

Now the code starts to talk to the SD card, it tries to initialize the card and find a FAT16/FAT32 partition.

Next it will try to make a logfile. We do a little tricky thing here, we basically want the files to be called something like **LOGGERnn.csv** where **nn** is a number. By starting out trying to create **LOGGER00.CSV** and incrementing every time when the file already exists, until we get to **LOGGER99.csv**, we basically make a new file every time the Arduino starts up

To create a file, we use some Unix style command flags which you can see in the `logfile.open()` procedure. **FILE_WRITE** means to create the file and write data to it.

Assuming we managed to create a file successfully, we print out the name to the Serial port.

```
  Wire.begin();
  if (!RTC.begin()) {
    logfile.println("RTC failed");
#if ECHO_TO_SERIAL
    Serial.println("RTC failed");
#endif  //ECHO_TO_SERIAL
  }


  logfile.println("millis,time,light,temp");
#if ECHO_TO_SERIAL
  Serial.println("millis,time,light,temp");
#if ECHO_TO_SERIAL// attempt to write out the header to the file
  if (logfile.writeError || !logfile.sync()) {
    error("write header");
  }

  pinMode(redLEDpin, OUTPUT);
  pinMode(greenLEDpin, OUTPUT);

   // If you want to set the aref to something other than 5v
  //analogReference(EXTERNAL);
}
```

OK we're wrapping up here. Now we kick off the RTC by initializing the Wire library and poking the RTC to see if its alive.

Then we print the header. The header is the first line of the file and helps your spreadsheet or math program identify whats coming up next. The data is in CSV (comma separated value) format so the header is too: "millis,time,light,temp" the first item **millis** is milliseconds since the Arduino started, **time** is the time and date from the RTC, **light** is the data from the CdS cell and **temp** is the temperature read.

You'll notice that right after each call to **logfile.print()** we have #if ECHO_TO_SERIAL and a matching **Serial.print()** call followed by a #if ECHO_TO_SERIAL this is that debugging output we mentioned earlier. The **logfile.print()** call is what writes data to our file on the SD card, it works pretty much the same as the **Serial** version. If you set **ECHO_TO_SERIAL** to be **0** up top, you won't see the written data printed to the Serial terminal.

Finally, we set the two LED pins to be outputs so we can use them to communicate with the user. There is a commented-out line where we set the analog reference voltage. This code assumes that you will be using the 'default' reference which is the VCC voltage for the chip - on a classic Arduino this is 5.0V. You can get better precision sometimes by lowering the reference. However we're going to keep this simple for now! Later on, you may want to experiment with it.

Main loop

Now we're onto the loop, the loop basically does the following over and over:

1. Wait until its time for the next reading (say once a second - depends on what we defined)
2. Ask for the current time and date froom the RTC
3. Log the time and date to the SD card
4. Read the photocell and temperature sensor
5. Log those readings to the SD card
6. Sync data to the card if its time

Timestamping

Lets look at the first section:

```
void loop(void)
{
  DateTime now;

  // delay for the amount of time we want between readings
  delay((LOG_INTERVAL -1) - (millis() % LOG_INTERVAL));

  digitalWrite(greenLEDpin, HIGH);

  // log milliseconds since starting
  uint32_t m = millis();
  logfile.print(m);           // milliseconds since start
  logfile.print(", ");
#if ECHO_TO_SERIAL
  Serial.print(m);         // milliseconds since start
  Serial.print(", ");
#endif

  // fetch the time
  now = RTC.now();
  // log time
  logfile.print(now.get()); // seconds since 2000
  logfile.print(", ");
  logfile.print(now.year(), DEC);
  logfile.print("/");
  logfile.print(now.month(), DEC);
  logfile.print("/");
  logfile.print(now.day(), DEC);
  logfile.print(" ");
  logfile.print(now.hour(), DEC);
  logfile.print(":");
  logfile.print(now.minute(), DEC);
  logfile.print(":");
  logfile.print(now.second(), DEC);
#if ECHO_TO_SERIAL
  Serial.print(now.get()); // seconds since 2000
  Serial.print(", ");
  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);
#endif //ECHO_TO_SERIAL
```

The first important thing is the **delay()** call, this is what makes the Arduino wait around until its time to take another reading. If you recall we **#defined** the delay between readings to be 1000 millseconds (1 second). By having more delay between readings we can use less power and not fill the card as fast. Its basically a tradeoff how often you want to read data but for basic long term logging, taking data every second or so will result in plenty of data!

Then we turn the green LED on, this is useful to tell us that yes we're reading/writing data now.

Next we call **millis()** to get the 'time since arduino turned on' and log that to the card. It can be handy to have - especially if you end up not using the RTC.

Then the familiar **RTC.now()** call to get a snapshot of the time. Once we have that, we write a timestamp (seconods since 2000) as well as the date in **YY/MM/DD HH:MM:SS** time format which can easily be recognized by a spreadsheet. We have both because the nice thing about a timestamp is that its going to montonically increase and the nice thing about printed out date is its human readable

Log sensor data

Next is the sensor logging code

```
int photocellReading = analogRead(photocellPin);
  delay(10);
  int tempReading = analogRead(tempPin);

  // converting that reading to voltage, for 3.3v arduino use 3.3
  float voltage = (tempReading * 5.0) / 1024.0;
  float temperatureC = (voltage - 0.5) * 100.0 ;
  float temperatureF = (temperatureC * 9.0 / 5.0) + 32.0;

  logfile.print(", ");
  logfile.print(photocellReading);
  logfile.print(", ");
  logfile.println(temperatureF);
#if ECHO_TO_SERIAL
  Serial.print(", ");
  Serial.print(photocellReading);
  Serial.print(", ");
  Serial.println(temperatureF);
#endif //ECHO_TO_SERIAL

  digitalWrite(greenLEDpin, LOW);
}
```

This code is pretty straight forward, the processing code is snagged from our earlier tutorial. Then we just **print()** it to the card with a comma seperating the two

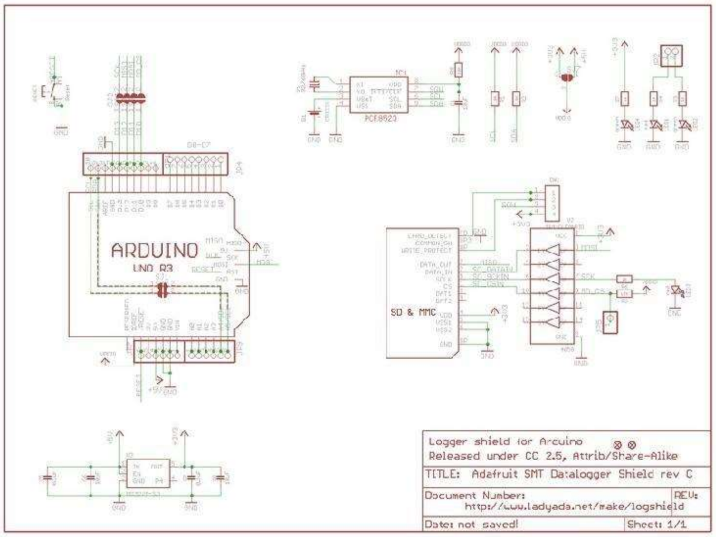
We finish up by turning the green LED off

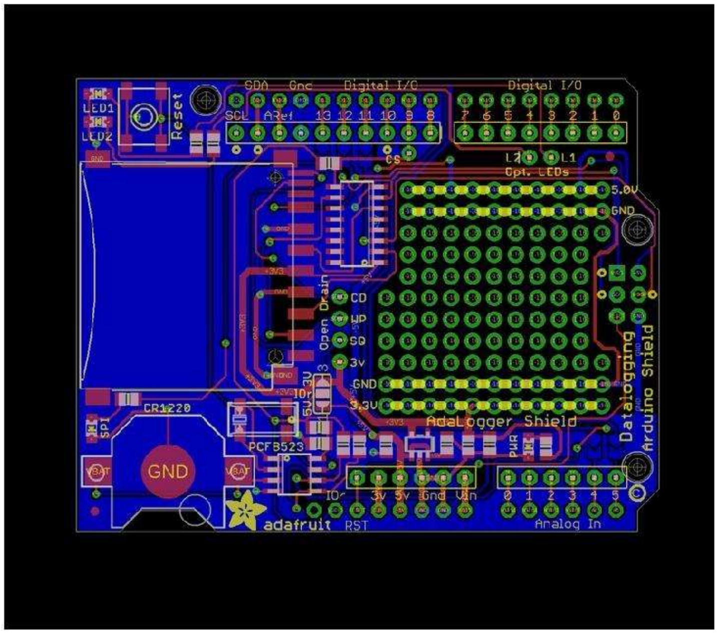
Downloads

Files

- EagleCAD PCB files on GitHub (https://adafru.it/rej)
- Fritzing object in Adafruit Fritzing library (https://adafru.it/aP3)

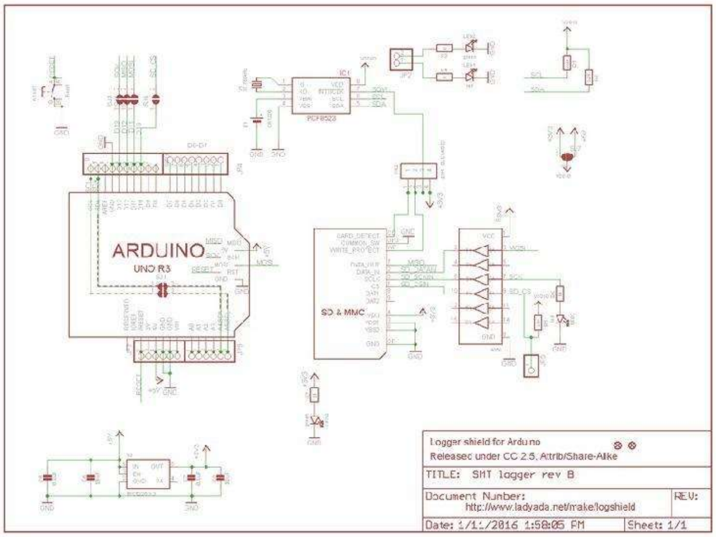
Revision C Schematics & Fabrication Print

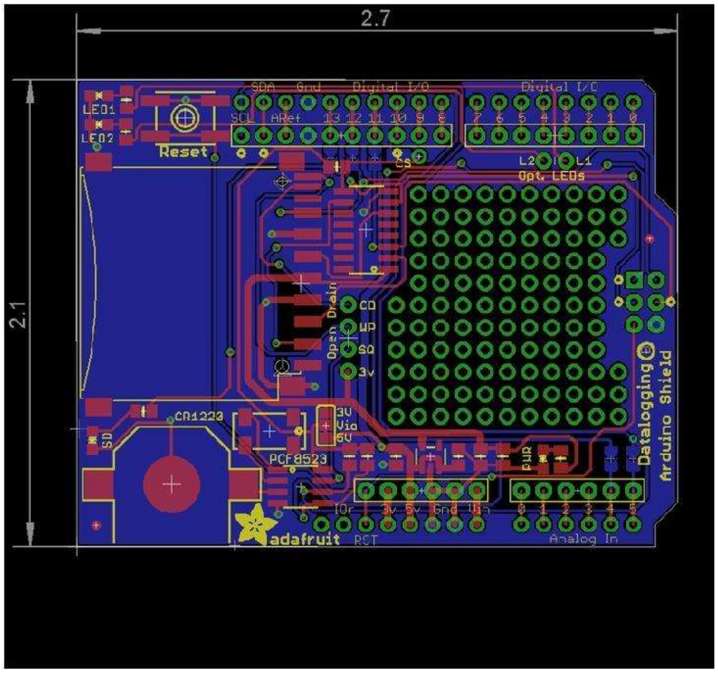

ARDUINO
UNO R3
PCF8523
SD & MMC
D8-D7
Logger shield for Arcuino
Released under CC 2.5, Attrib/Share-Alike
TITLE: Adafruit SMT Da:alogger Shielc rev C
Document Number:
http://www.ladyada.net/make/logshield
REV:
Date: not saved!
Sheet: 1/1



Revision B Schematics

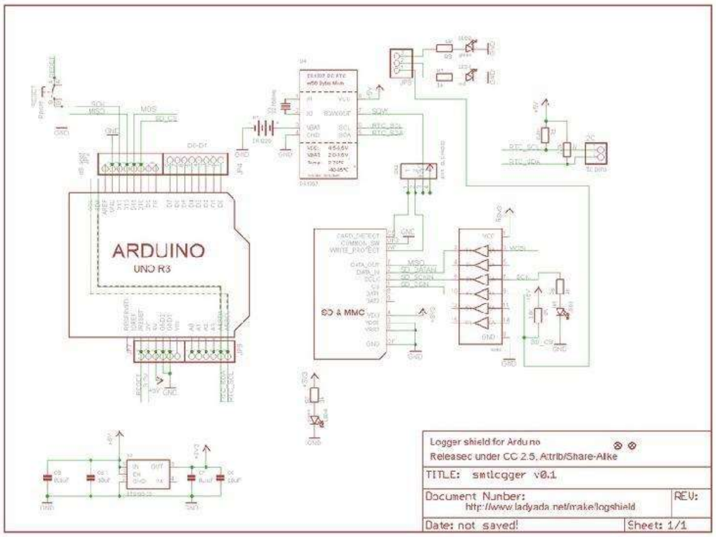
click to enlarge


ARDUINO
UNO R3
PCF8523
SD & MMC
Logger shield for Arduino
Released under CC 2.5, Attrib/Share-Alike
TITLE: SHT logger rev B
Document Number:
http://www.ladyada.net/make/logshield
REV:
Date: 1/11/2016 1:58:05 PM
Sheet: 1/1



Original Version Schematics

Click to enlarge


ARDUINO
UNO R3
SD & MMC
Logger shield for Arduino
Released under CC 2.5, Attrib/Share-Alike
TITLE: smtlogger v0.1
Document Number:
http://www.ladyada.net/make/logshield
REV:
Date: not saved!
Sheet: 1/1

10.9 Appendix I: K33 ICB 10% CO2 Sensor Datasheet

CO2Meter.com
CO2 Measurement Specialists

Product Manual

CO_2 Engine™- ICB

Sensor module for bio applications

SE-0025 10% CO2 Sensor Module
SE-0026 30% CO2 Sensor Module
CM-0058 10% CO2 Development Kit
CM-0027 30% CO2 Development Kit

General

- ***CO_2 Engine™ ICB*** is targeted for biological applications with required measurement range 0 to up to $30\%_{vol}$ CO_2. This document contains a description of the programming I/O and precise measurements for OEM applications.
- The ***CO_2 Engine™ ICB*** is built on the ***CO_2 Engine™ K33 platform.*** This platform is designed to be an OEM module or as a standalone CO2 transmitter/switch module.
- The ***CO_2 Engine™ ICB*** has the same dimension and attachment points as K30 platform based sensors.

Connection to host system alternatives

(See electrical specifications in the table "Terminal description" below)

Connection alternative A.

CO₂ Engine™ ICB is built in into the customer's system by connection via JP5. I^2C communication is used to read measured data from the sensor. Detailed description of I^2C communication with useful examples and troubleshooting can be found in "I2C comm guide 2_14.pdf"

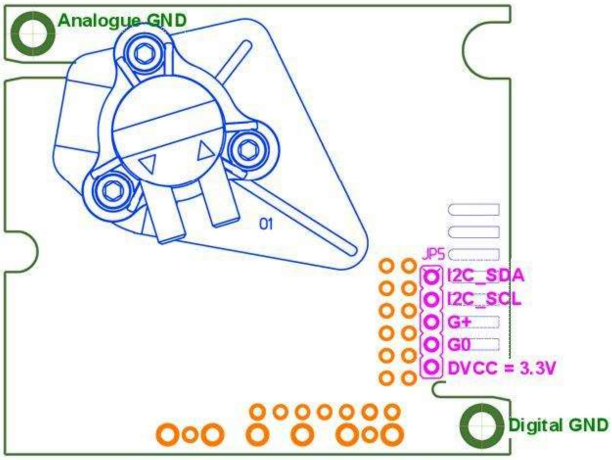


Figure 2. ***CO₂ Engine™ ICB*** *Possible connection terminals for reading via I^2C.*

Note: Both Digital GND and Analog GND are connected to G0 internally.

Connection alternative B.

CO_2 Engine™ *ICB* is built in into the customer's system by connection via JP1 or some part of it. UART with Modbus protocol communication is used to read measured data from the sensor. ***CO_2 Engine™*** *ICB* shares specification and Modbus register map with CO2 Engine K30 sensor family. Specification can be found in "ModBus on CO2 Engine K30 rev1_07.pdf" and "K30 ModBus map rev1_01.xls"

View from component side:

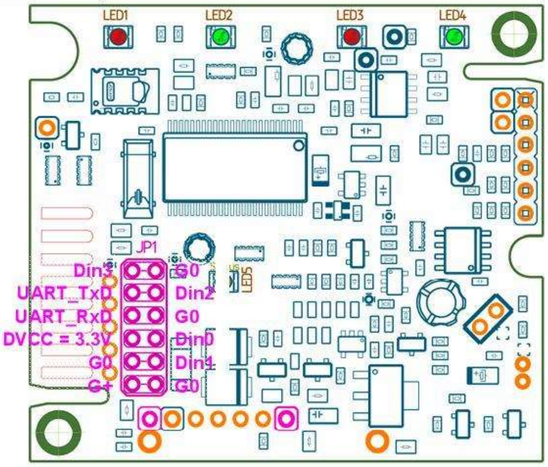


Figure 3a. ***CO_2 Engine™ ICB*** *Possible connection terminals for reading via UART.*

Note: LEDs are optional.

View from OBA side:

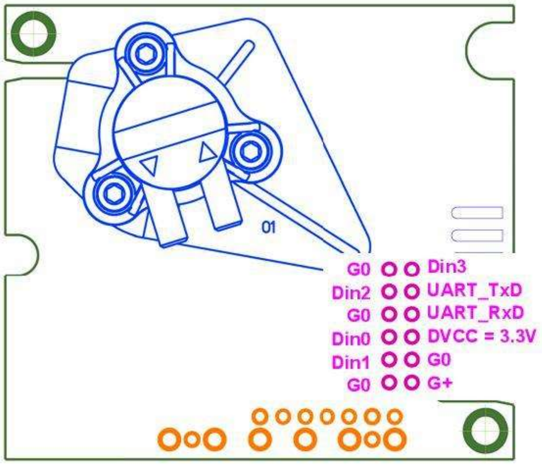


Figure 3b. ***CO_2 Engine™ ICB*** *Possible connection terminals for reading via UART.*

Connection alternative C.

CO₂ Engine™ ICB is built in into the customer's system by connection via terminals. Signal lines on these terminals are protected and long wires may be used for connection to the host system.

5.08 mm pitch:

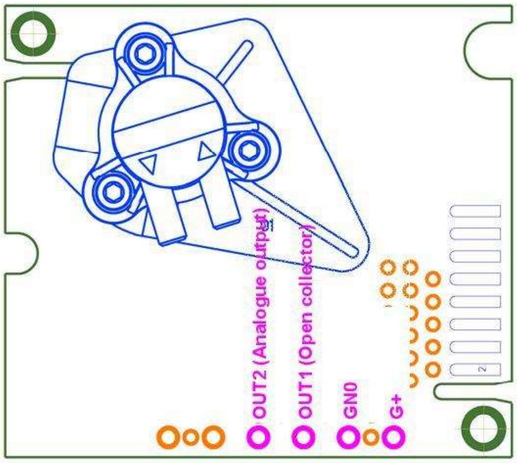


Figure 4a. ***CO₂ Engine™ ICB*** *Possible connection terminals for connection by long wires.*

2 mm pitch:

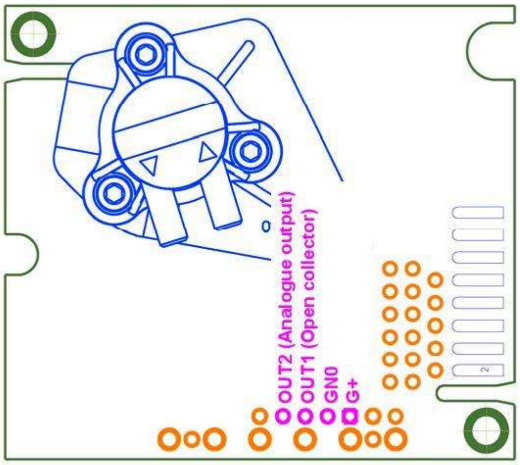


Figure 4b. ***CO₂ Engine™ ICB*** *Possible connection terminals for connection by long wires.*

Note: OUT1, open collector is configured to provide PWM signal, see specification below.

Connection alternative D.

Combination of alternatives B and C. It's possible to use both UART and OUT1 at the same time. In the same way it's possible to use alternatives A and C, I2C and OUT1 at the same time.

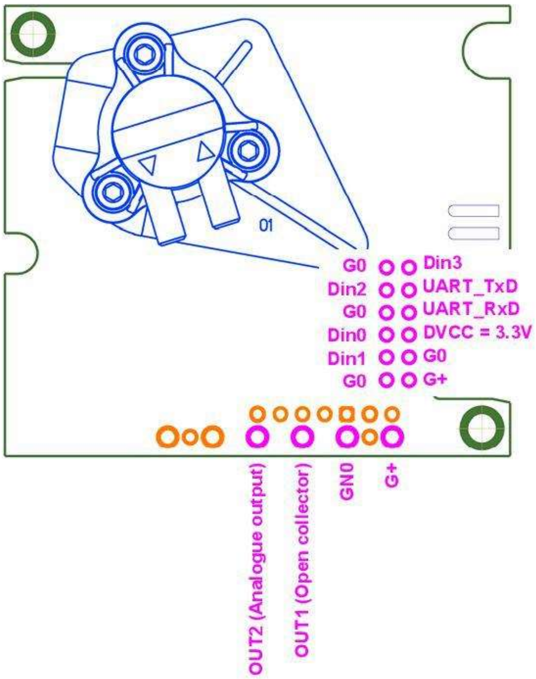


Figure 5. ***CO$_2$ Engine™ ICB*** *Possible connection terminals for connection by long wires and UART at the same time.*

Diffusion or tube IN/OUT alternatives

CO_2 Engine™ *ICB* can be supplied in diffusion modification with or without O-ring.

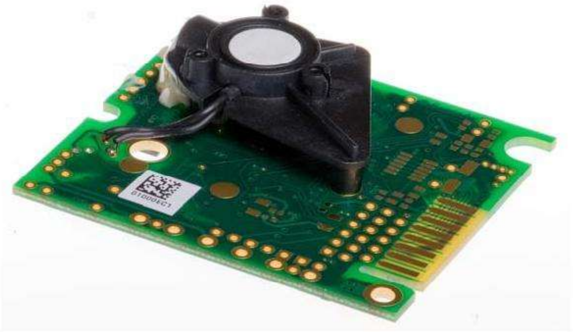


Figure 6. ***CO_2 Engine™ ICB*** *diffusion model.*

CO_2 Engine™ *ICB* can be supplied in tube in/out modification with different orientation of tube attachment head in steps of 120 degrees.

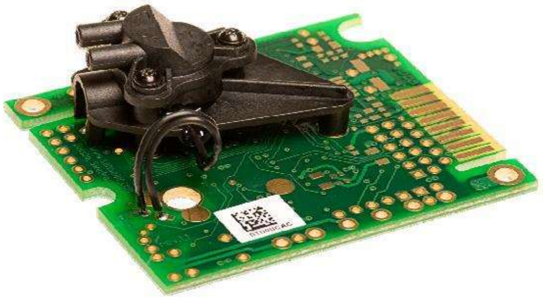


Figure 7. ***CO_2 Engine™ ICB*** *tube IN/OUT model.*

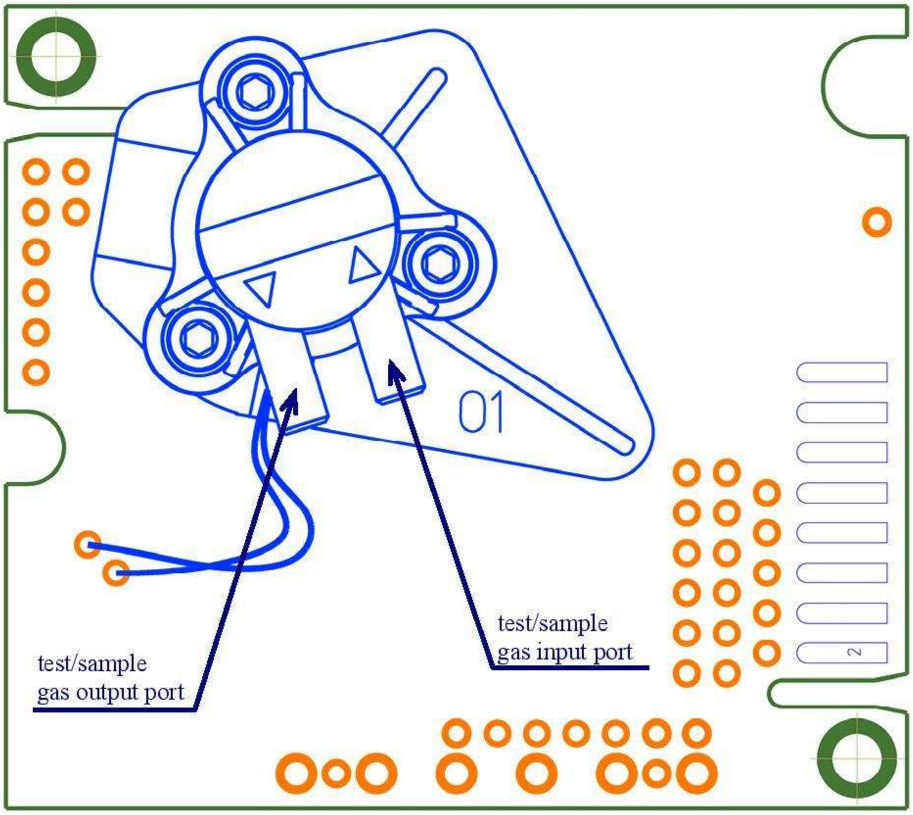


Figure 8a . ***CO$_2$Engine™ ICB*** *Test/sample gas ports.*

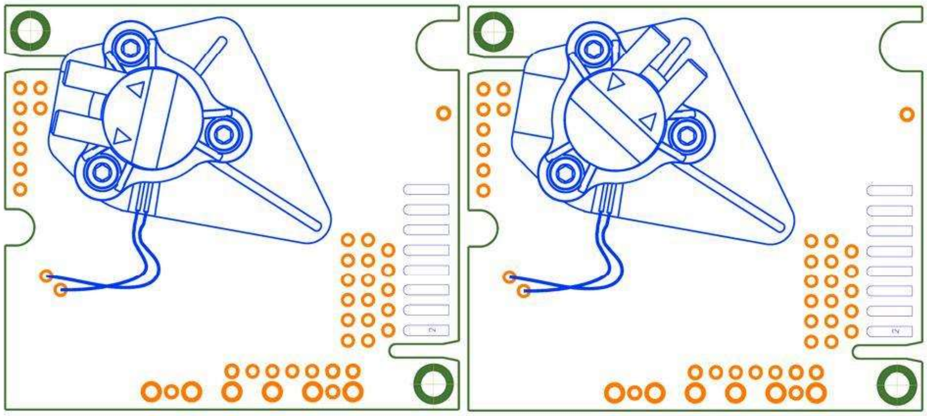


Figure 8b . ***CO_2Engine™ ICB*** *Possible test/sample gas ports installations.*

Terminal description

The table below specifies terminals and I/O options available in the general **K33** platform *(see also the alternative connection pictures above).*

Functional group	Descriptions and ratings
Power supply (all connection alternatives)	
G+ referred to G0	Power supply plus terminal Protected by series 3.3R resistor and zener diode Absolute maximum ratings 5 to 14V, stabilized to within 10%
G0	Power supply minus terminal Sensor's reference (ground) terminal
DVCC = 3.3V	Output from sensor's digital voltage regulator. Series resistance 10 R Available current 12mA Voltage tolerance (unloaded) +-3% max (+-0.75% typ) Output may be used to power circuit (microcontroller) in host system or to power logical level converter if master processor runs at 5V supply voltage.
Communication	
UART (UART_TxD, UART_RxD)	CMOS physical layer, ModBus communication protocol. (refer "ModBus on CO2 Engine K30 rev1_07.pdf" or later version for details) UART_RxD line is configured as digital input. Input high level is 2.1V min Input low level is 0.8V max UART_TxD line is configured as digital output. Output high level is 2.3V (assuming 3.3V DVCC) min. Output low level is 0.75V max UART_RxD input is pulled up to DVCC = 3.3V by 56 kOhm UART_TxD output is pulled up to DVCC = 3.3V by 56 kOhm ABSOLUTE MAX RATING G0-0.5V ..... DVCC + 0.5V
I2C extension. (I2C_SCL, I2C_SDA)	Pull-up to DVCC = 3.3V. (refer "I2C comm guide rev2_00 DRAFT.pdf" or later version for details) ABSOLUTE MAX RATING G0-0.5V ..... DVCC + 0.5V
Outputs	
OUT1, OC (Open collector)	Digital output, Open collector Series resistance 120 R Max sink current 40mA May be configured as

	1. Alarm indication output 2. PWM output, 10 (alt. 12 to 16) bit resolution. Period 1 .. 1000 msec 3. Pulse length proportional to measured CO2 value.
OUT2	Analog output 0..5V Buffered linear output 0..4 or 1..4VDC or 0..5V or 1..5V, depending on specified power supply and sensor configuration. $R_{OUT} < 100\ \Omega$, $R_{LOAD} > 5\ k\Omega$ **Load to ground only!** Resolution 5mV
RELAY (RelayPoleNC RelayPoleCom RelayPoleNO)	RELAY It's not a standard option. Maximum switching capability 1A/50VAC/24VDC
Digital I/Os, used as Inputs in standard configuration. May be implemented as jumper field	
Din0 Din1 Din2	Digital switch inputs in standard configuration, Pull-up 56k to DVCC 3.3V. Driving it Low or connecting to G0 activates input. Pull-up resistance is decreased to 4..10k during read of input or jumper. Advantages are lower consumption most of the time the input/jumper is kept low and larger current for jumpers read in order to provide cleaning of the contact. Can be used for zero or background calibration forcing.
Din3	R/T control line for UART connection to RS485 driver.

Table I. I/O notations used in this document for the K33 platform with some descriptions and ratings. Please, beware of ***the red colored texts that pinpoint important features*** *for the system integration!*

Mechanical drawings

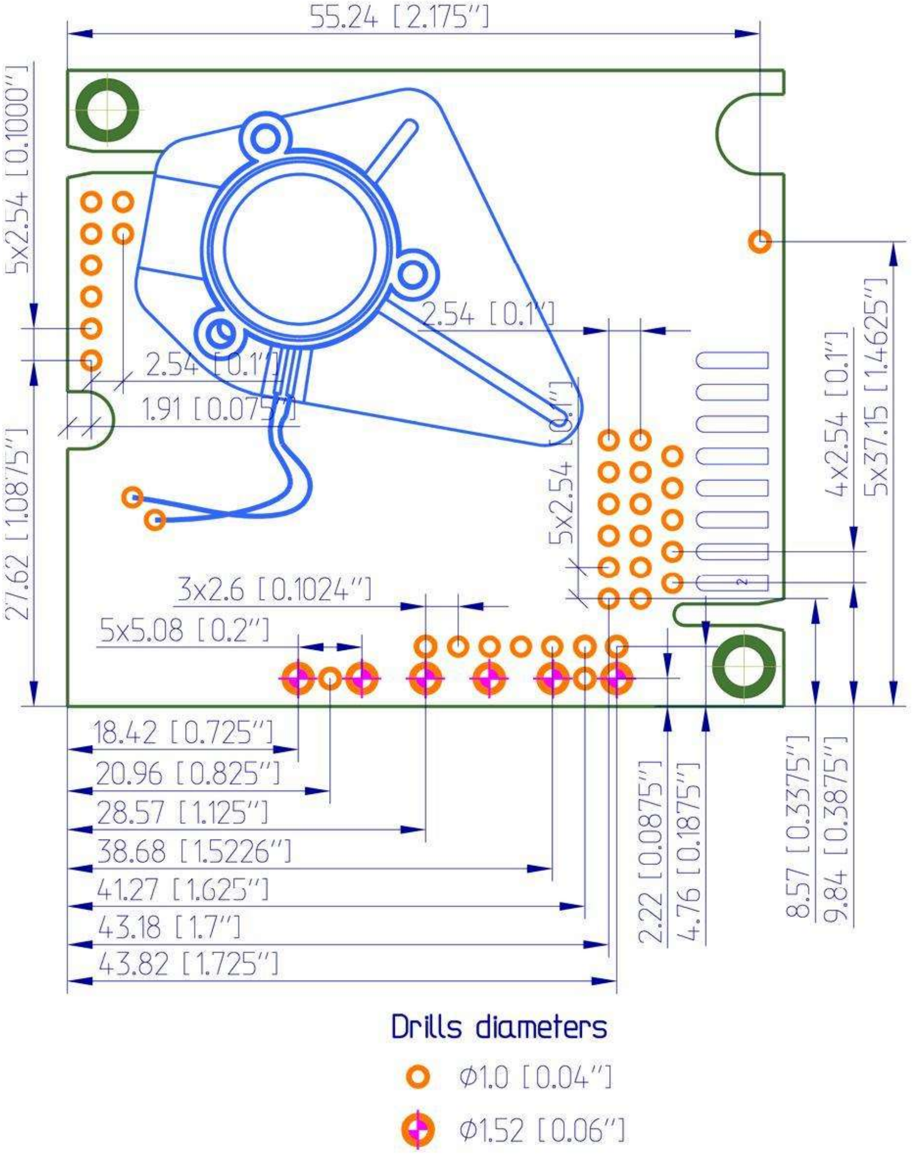


Figure 9 . ***CO$_2$Engine™ ICB*** *mechanical drawing. Hole/contacts positions.*

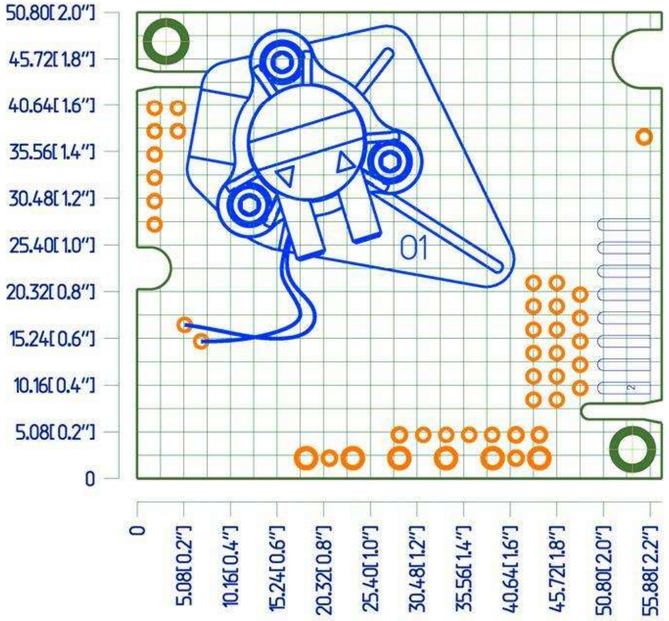


Figure 10a . ***CO_2Engine™ ICB*** *OBA position. Tube IN/OUT model*

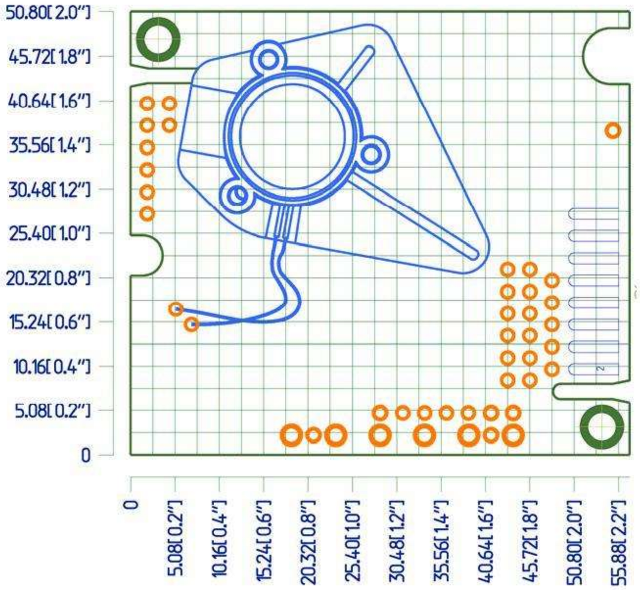


Figure 10b . ***CO_2Engine™ ICB*** *OBA position. Diffusion model..*

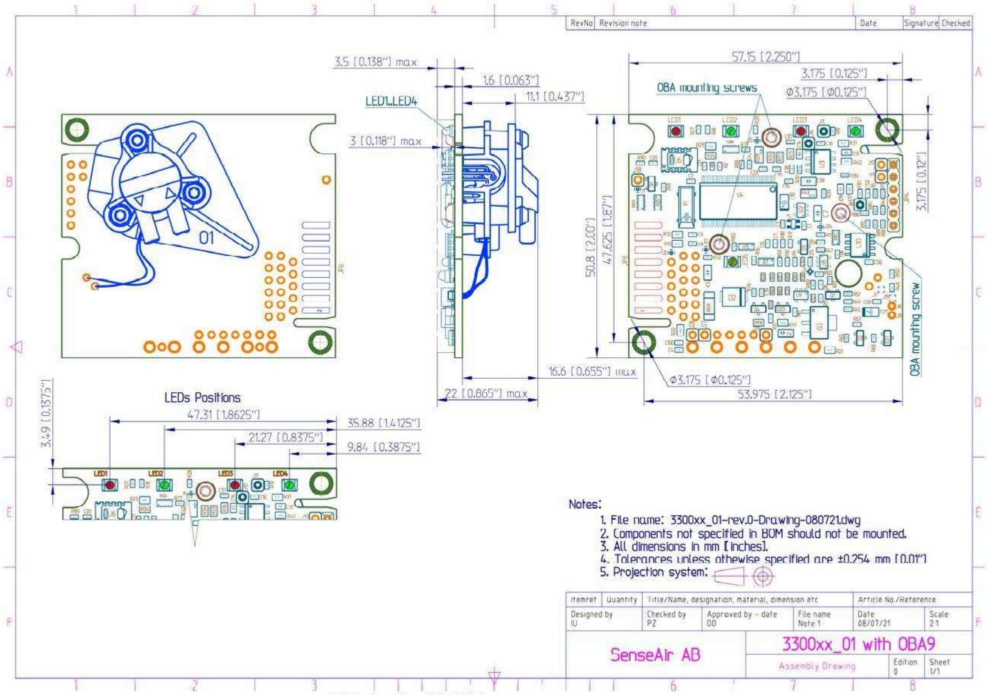


Figure 11 . ***CO_2Engine™ ICB*** *mechanical drawing. Tube IN/OUT model*

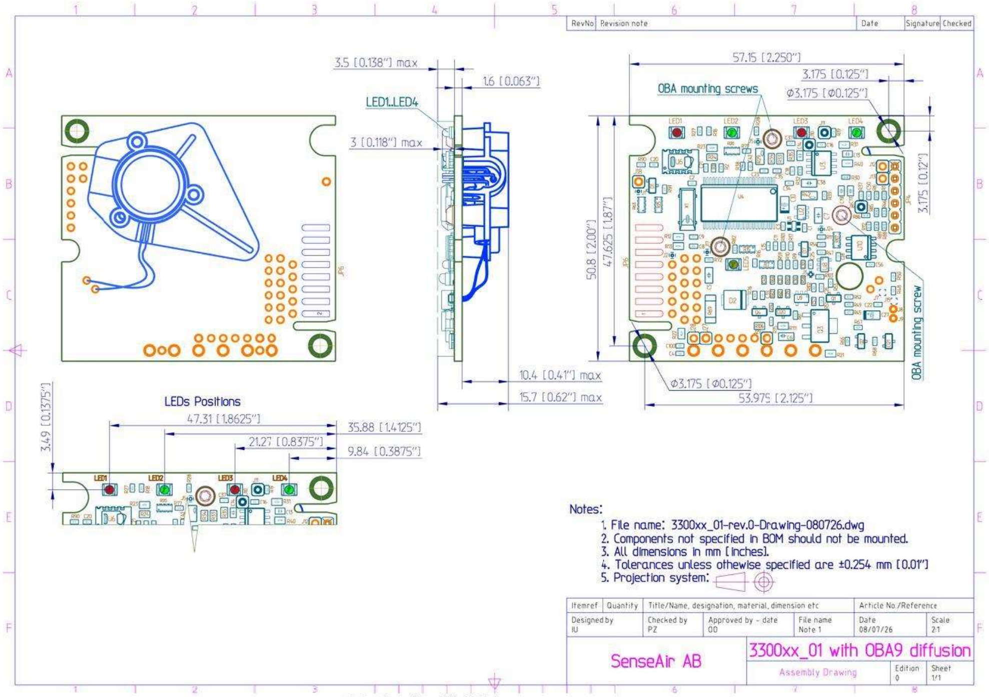


Figure 12 . ***CO_2Engine™ ICB*** *mechanical drawing. Diffusion model.*

Ground / Shield attachments

Both Analog ground (AGND) and digital ground (DGND) are connected internally to the G0 terminal of the sensor. AGND is connected to the most sensitive analogue part of the sensor and DGND is connected to the digital part of the sensor.

Do NOT connect AGND and DGND together externally to sensor!

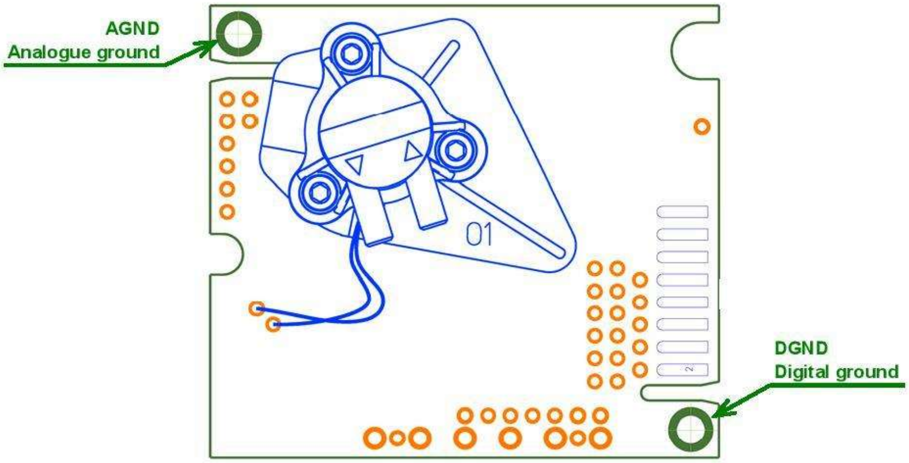


Figure 13 . ***CO_2Engine™ ICB*** *ground / shield attachment*

Maintenance

When used in environments where the built-in self-correcting ***ABC*** **algorithm** can be enabled the ***CO_2Engine™ ICB*** is basically maintenance free. Since the ABC algorithm can not be used in all applications it is disabled in sensors default appearance.
Discuss your application with SenseAir in order to get advice for a proper calibration strategy.

When checking the sensor accuracy, PLEASE NOTE that the sensor accuracy is defined at continuous operation with enabled ABC algorithm (at least 3 weeks after installation) or after zero/background calibration.

Calibration

When enabled the ***ABC*** **algorithm** *(Automatic Baseline Correction)* constantly keeps track of the sensor's lowest reading over a 7,5 days interval and slowly corrects for any long-term drift detected as compared to the expected fresh air value of $0.04\%_{vol}$ CO_2.

Rough handling and transportation might result in a reduction of sensor reading accuracy. If the ABC algorithm is enabled it will tune the readings back to the correct numbers. The default "tuning speed" is however limited. This limit is application specific. In case that the ABC function is disabled (default appearance) or one cannot wait for the ABC algorithm to cure any calibration offset, two switch inputs Din1 and Din2 are defined for the operator to select one out of two prepared calibration codes. If Din1 is shorted to ground, for a minimum time of 8 seconds, the internal calibration code **bCAL** *(background calibration)* is executed, in which case it is assumed that the sensor is operating in a fresh air environment (400 ppm CO_2). If Din2 is shorted instead, for a minimum time of 8 seconds, the alternative operation code **CAL** *(zero calibration)* is executed in which case the sensor must be purged by some gas mixture free from CO_2 (i.e. Nitrogen or Soda Lime CO_2 scrubbed air). If unsuccessful, please wait at least 10 seconds before repeating the procedure again. Make sure that the sensor environment is steady and calm!

Input **Switch Terminal** *(normally open)*	**Default function** *(when closed for minimum 8 seconds)*
Din1	**bCAL** (background calibration) assuming 400 ppm CO_2 sensor exposure
Din2	**CAL** (zero calibration) assuming 0 ppm CO_2 sensor exposure

Table II. Switch input default configurations for ***CO_2Engine™ ICB***

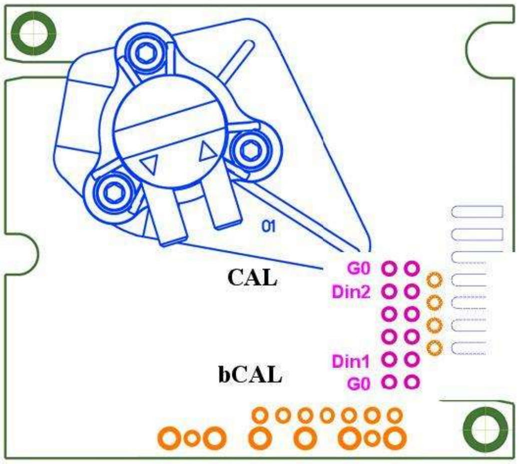
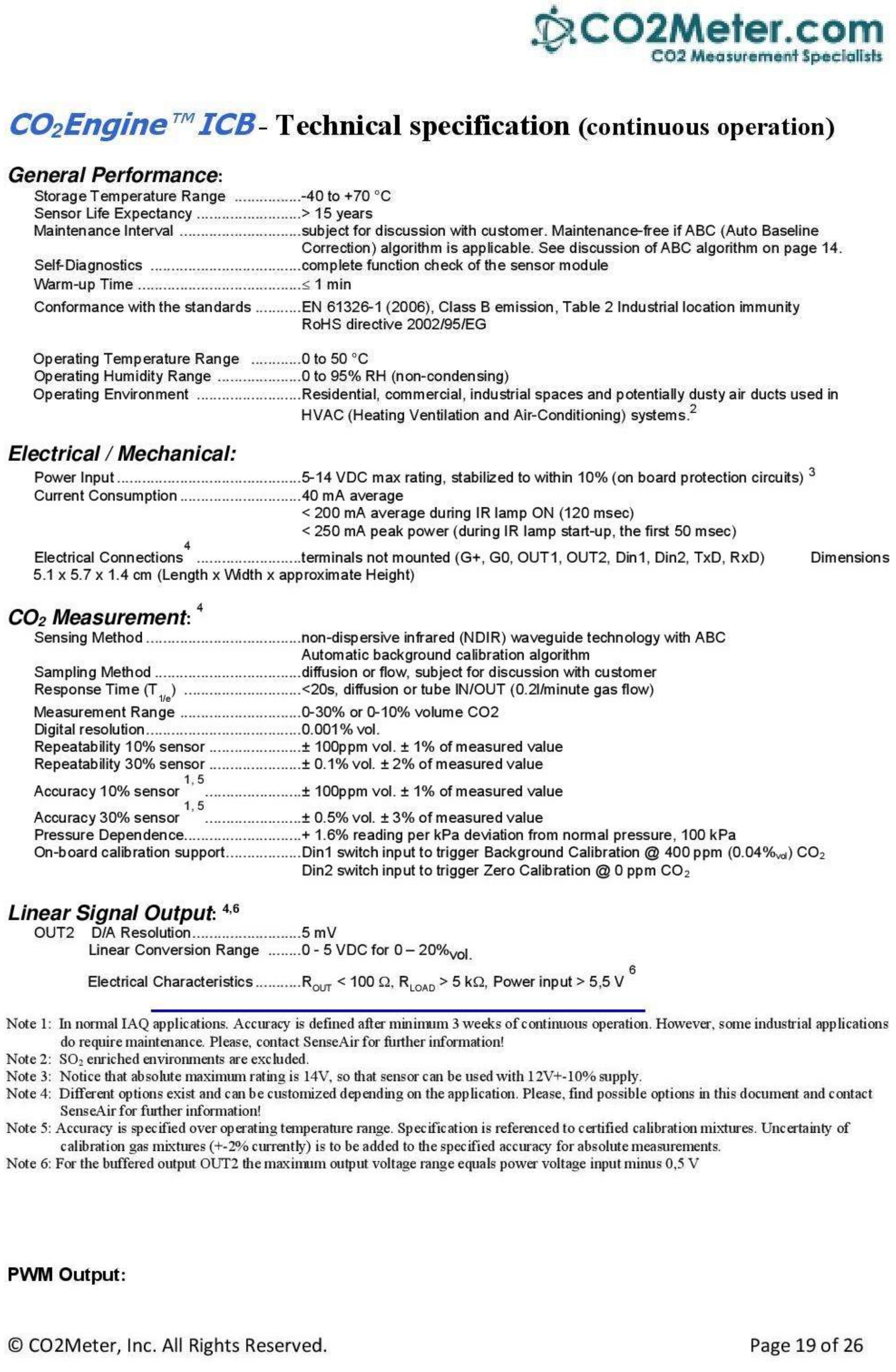


Figure 14. ***CO_2 Engine™ ICB*** *calibration jumpers.*



CO2Meter.com
CO2 Measurement Specialists

CO₂Engine™ ICB - Technical specification (continuous operation)

General Performance:

Storage Temperature Range	-40 to +70 °C
Sensor Life Expectancy	> 15 years
Maintenance Interval	subject for discussion with customer. Maintenance-free if ABC (Auto Baseline Correction) algorithm is applicable. See discussion of ABC algorithm on page 14.
Self-Diagnostics	complete function check of the sensor module
Warm-up Time	≤ 1 min
Conformance with the standards	EN 61326-1 (2006), Class B emission, Table 2 Industrial location immunity RoHS directive 2002/95/EG
Operating Temperature Range	0 to 50 °C
Operating Humidity Range	0 to 95% RH (non-condensing)
Operating Environment	Residential, commercial, industrial spaces and potentially dusty air ducts used in HVAC (Heating Ventilation and Air-Conditioning) systems.[2]

Electrical / Mechanical:

Power Input	5-14 VDC max rating, stabilized to within 10% (on board protection circuits) [3]
Current Consumption	40 mA average < 200 mA average during IR lamp ON (120 msec) < 250 mA peak power (during IR lamp start-up, the first 50 msec)
Electrical Connections [4]	terminals not mounted (G+, G0, OUT1, OUT2, Din1, Din2, TxD, RxD)
Dimensions	5.1 x 5.7 x 1.4 cm (Length x Width x approximate Height)

CO₂ Measurement: [4]

Sensing Method	non-dispersive infrared (NDIR) waveguide technology with ABC Automatic background calibration algorithm
Sampling Method	diffusion or flow, subject for discussion with customer
Response Time ($T_{1/e}$)	<20s, diffusion or tube IN/OUT (0.2l/minute gas flow)
Measurement Range	0-30% or 0-10% volume CO2
Digital resolution	0.001% vol.
Repeatability 10% sensor	± 100ppm vol. ± 1% of measured value
Repeatability 30% sensor	± 0.1% vol. ± 2% of measured value
Accuracy 10% sensor [1, 5]	± 100ppm vol. ± 1% of measured value
Accuracy 30% sensor [1, 5]	± 0.5% vol. ± 3% of measured value
Pressure Dependence	+ 1.6% reading per kPa deviation from normal pressure, 100 kPa
On-board calibration support	Din1 switch input to trigger Background Calibration @ 400 ppm (0.04%$_{vol}$) CO₂ Din2 switch input to trigger Zero Calibration @ 0 ppm CO₂

Linear Signal Output: [4,6]

OUT2	D/A Resolution	5 mV
	Linear Conversion Range	0 - 5 VDC for 0 – 20%$_{vol}$.
	Electrical Characteristics	R_{OUT} < 100 Ω, R_{LOAD} > 5 kΩ, Power input > 5,5 V [6]

Note 1: In normal IAQ applications. Accuracy is defined after minimum 3 weeks of continuous operation. However, some industrial applications do require maintenance. Please, contact SenseAir for further information!

Note 2: SO₂ enriched environments are excluded.

Note 3: Notice that absolute maximum rating is 14V, so that sensor can be used with 12V+-10% supply.

Note 4: Different options exist and can be customized depending on the application. Please, find possible options in this document and contact SenseAir for further information!

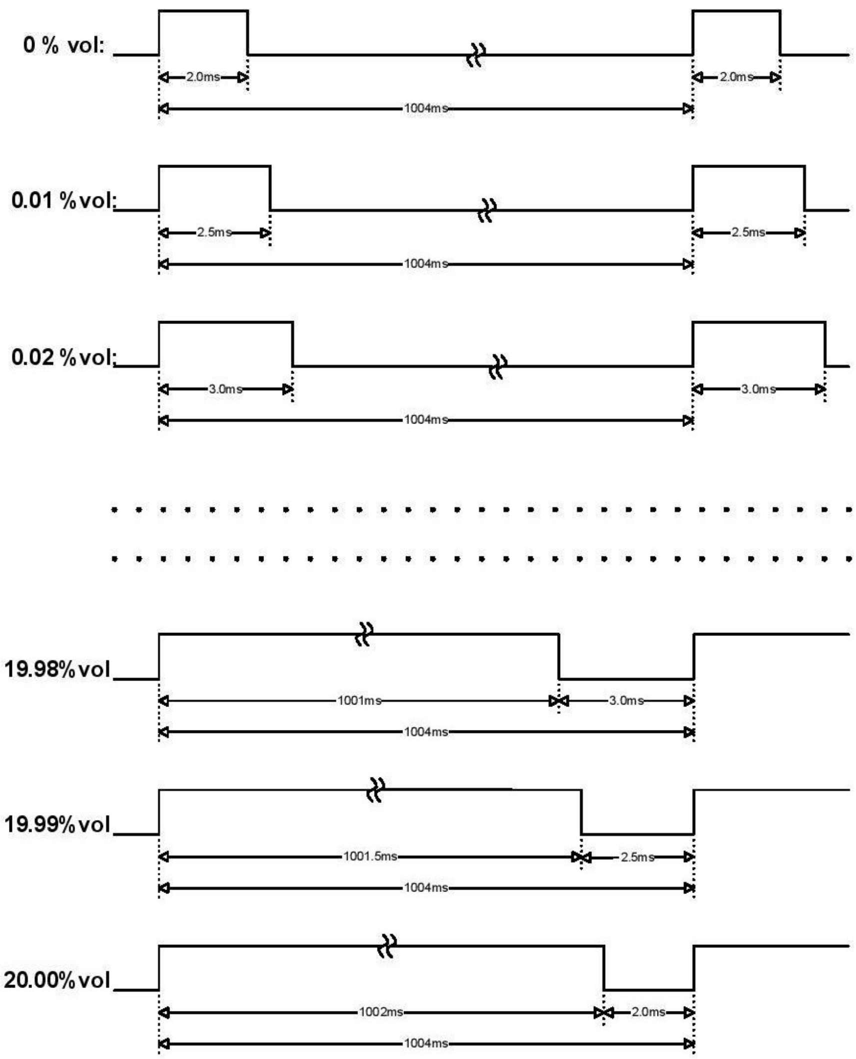
Note 5: Accuracy is specified over operating temperature range. Specification is referenced to certified calibration mixtures. Uncertainty of calibration gas mixtures (+-2% currently) is to be added to the specified accuracy for absolute measurements.

Note 6: For the buffered output OUT2 the maximum output voltage range equals power voltage input minus 0,5 V

PWM Output:

Electrical Characteristics	Open collector with series 120R resistor, 10kΩ pull-up resistor to protected power (+)
Minimum output concentration	$0\%_{vol}$
Output cycle period	1004ms
Output high level min duration	2.0ms (@ $0\%_{vol}$)
Output high level max duration	1002ms (@ $20\%_{vol.}$)
Resolution	0.5ms (@$0.01\%_{vol}$ = 100 ppm)

Sensor PWM output timing diagram



Materials

Component / coating	Material	Notes
PCB	FR4, Base laminate – copper clad glass base epoxy resin in accordance with IPC-4101/124	
Surface finish of unsoldered copper on PCB	Gold plated, ENIG, thickness 0.05 um min	
Solder mask	Liquid photo-image able	
OBA	LCP700	TBD, it is plastic from LCP family. It’s chemically inert plastic stable in most chemicals.
OBA coating	OBA is not coated	Absence of coating provides extra resistance of OBA against corrosive environments

Gases that may affect sensor’s operation

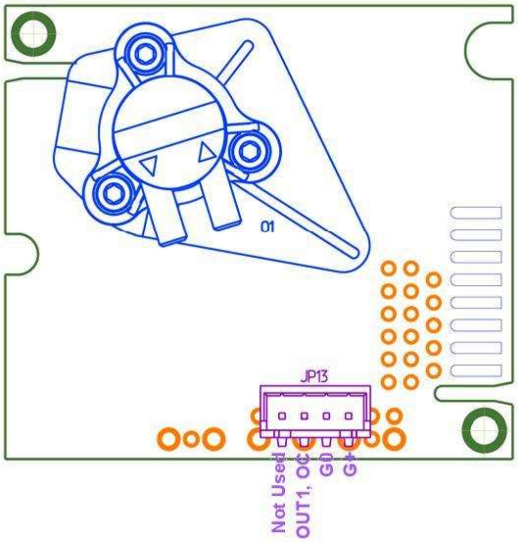
Since optical part has no reflective coating, stability of the sensor is governed by corrosion resistance of electronic assembly.
Corrosive environments containing but not limited by hydrogen sulfide, ammonia, ozone, sulphuric acid, sulfur dioxide should be avoided.

Use Ideas (connections)

Alternative E.

CO₂ Engine™ *ICB* is a stand-alone module connected to the host system by 3-wire interface (JP13 may be chosen to be 3 pole connector) with power supply and open collector output for alarm condition indication. Open collector output may drive LED or relay or buzzer in the case of alarm conditions.

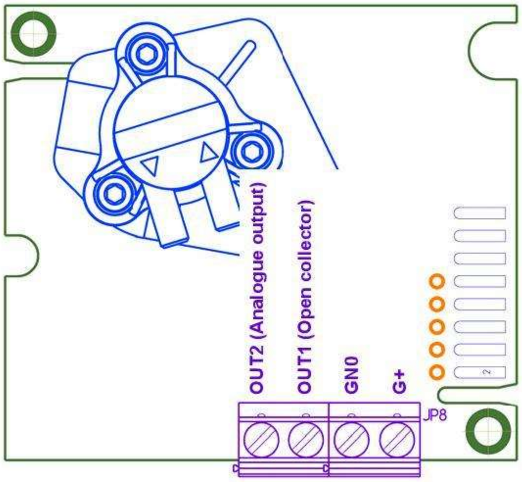
Open collector may provide PWM signal with duty cycle representing CO2 concentration



Alternative F.

CO₂ Engine™ *ICB* is a stand-alone module connected to the host system by 3-wire interface (JP8 may be chosen to be 3 pole terminal) with power supply and open collector output for alarm condition indication. Open collector output may drive LED or relay or buzzer in the case of alarm conditions.

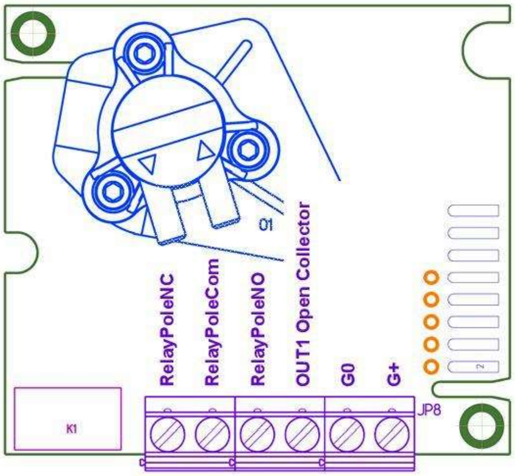
Open collector may provide PWM signal with duty cycle representing CO2 concentration



Alternative G.

CO₂ Engine™ ICB is a stand-alone module with open collector output and NC/NO relay

Open collector may provide PWM signal with duty cycle representing CO2 concentration



Frequently Asked Questions – Programming

1- If I send the stop command, Is it only the CO2 measurement process that is stopped, or is the sensor not going to respond to any other request except a start measurement request?

   When receiving a stop command sensor finishes its present task and goes into sleep mode (to minimize current consumption), in sleep mode sensor wakes up periodically (1s) and for example check the jumper, activity on the communication lines will also wake up the sensor (see I2C communication guide for details)

2- If I stop the measurement, are all the other functions still active, can I still read/write from the sensor? Can I still read the last CO2 value and temperature while the sensor is stopped?

   Yes, communication is possible after stop measurement.

3- When a start measurement CMD is sent, what is the delay to the effective start of the measurement process (how much time after is the sensor normally starting to hold the bus CLK)?

   Standard time to measurement is 1 minute (same for start measurement command and jumper set), measurement sequence takes ~20s

4- How quickly (immediately?) after the end of the bus hold can I send a stop command ? or is it better to leave the sensor running for a minimum delay?

   It is recommended to wait until after measurement sequence has finished and you have read new data before sending stop command (~80s)

5- Should the light stop flashing when the stop measurement command is sent?

   Not immediately, sensor will finish ongoing measurement sequence before going to sleep – stop measuring

Revision history

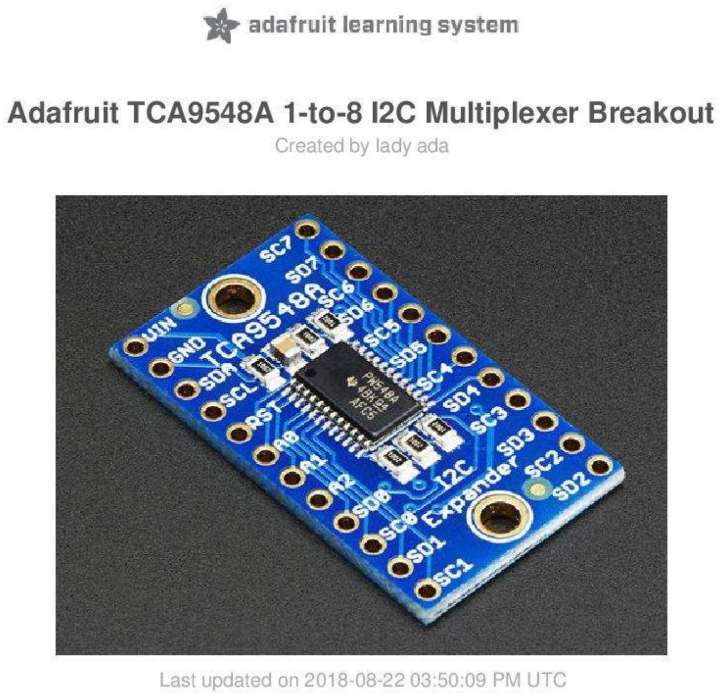
Edition	Date	By	Description
PA1	2008-05-01	PZ	First appearance
PA2	2008-07-21	IU	Correcting pictures
PA3	2008-07-27	IU	Correcting pictures
PA4	2008-08-07	PZ	Update with new pictures and PWM output.
PA5	2008-08-08	PZ	Added materials chapter.
PA6	2008-08-08	PZ	Convert CNT spec into ICB spec with corrections by HM
PA8	2009-11-02	JA	Correcting pictures
1.00	2011-07-04	LN	Correcting linear conversion range
1.01	2012-05-30	LN	CO2 measurement accuracy updated

For more information contact:

**CO2Meter.com
131 Business Center Drive
Ormond Beach, FL 32174 USA**

**(386) 256-4910 Support M-F 8:30am-5pm EST
(866) 422-2356 Fax
Support@CO2Meter.com**

10.10 Appendix J: Adafruit TCA9548A 1-to-8 I2C Multiplexer Breakout Datasheet

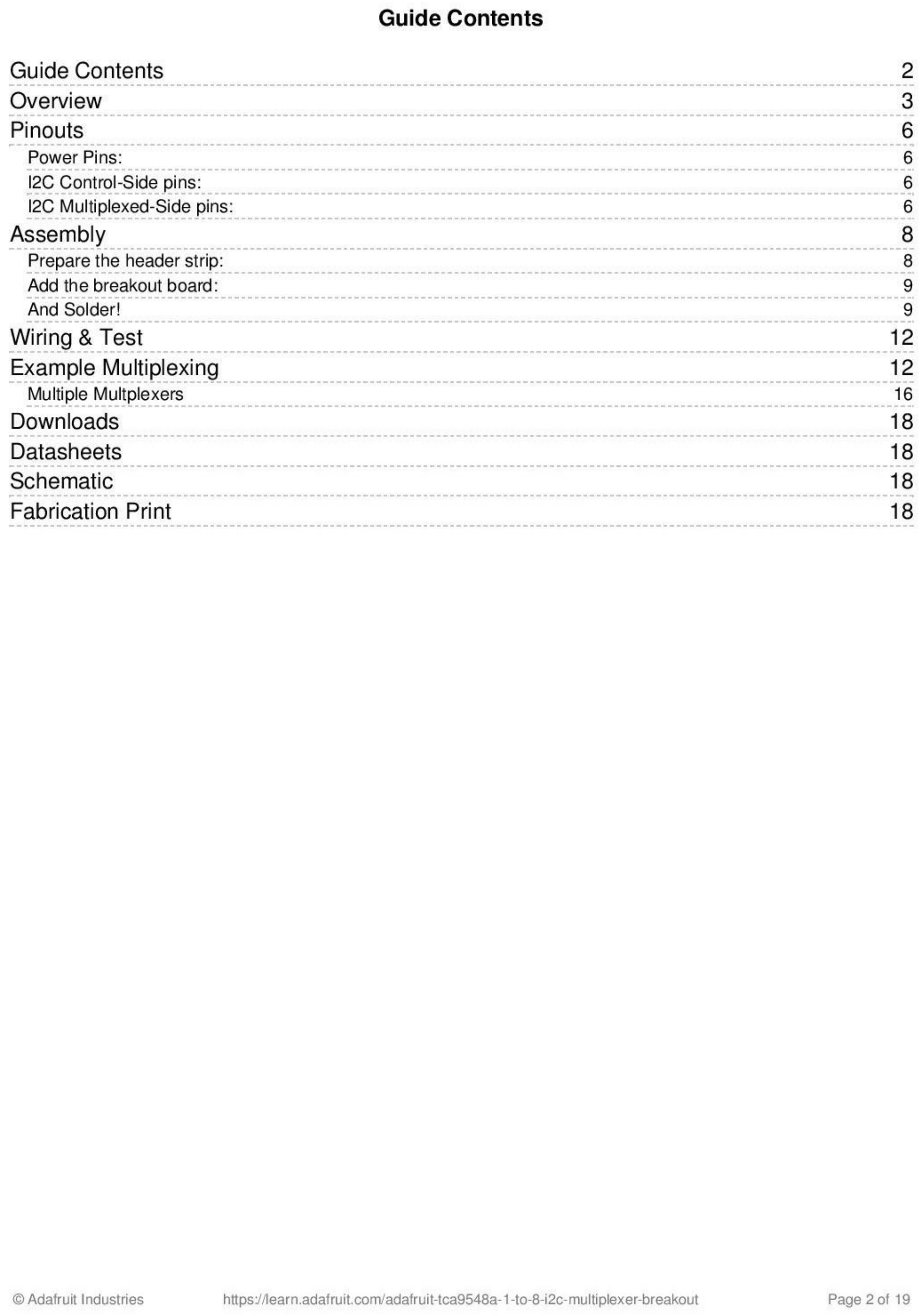


adafruit learning system

Adafruit TCA9548A 1-to-8 I2C Multiplexer Breakout

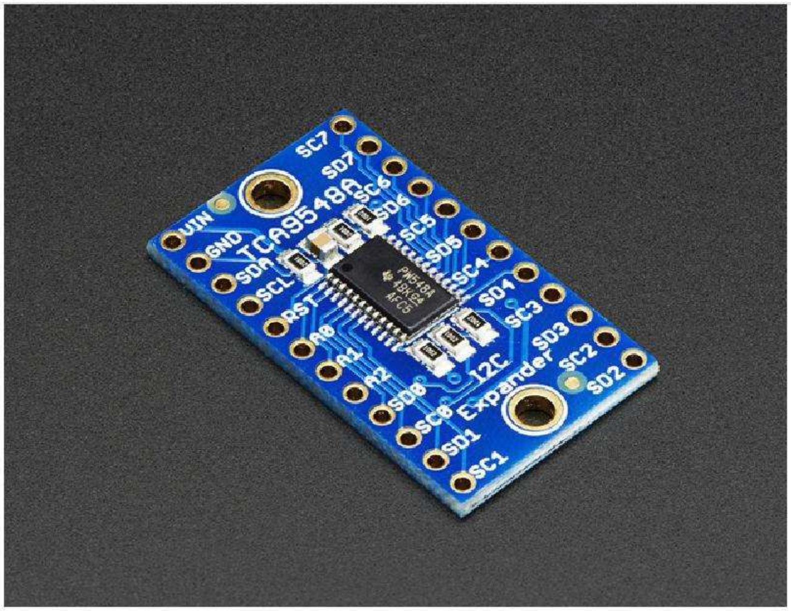
Created by lady ada

Last updated on 2018-08-22 03:50:09 PM UTC

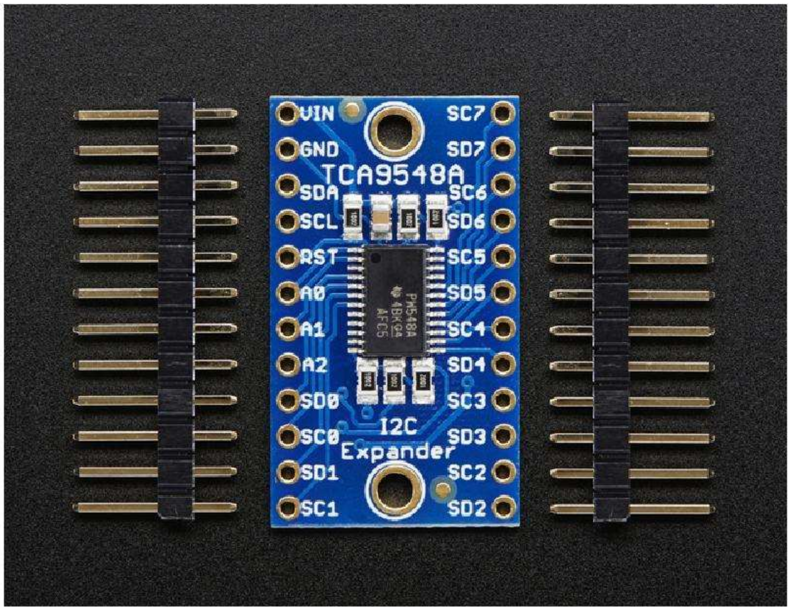


Guide Contents

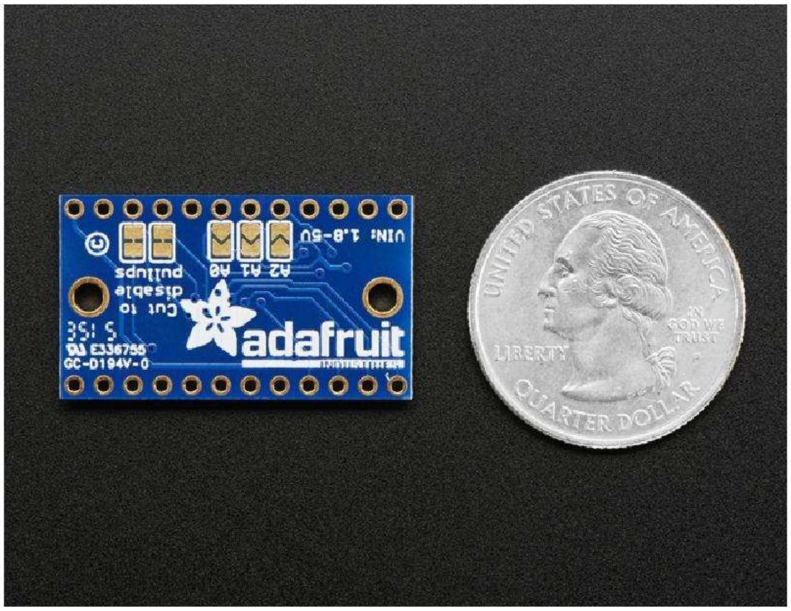
Overview



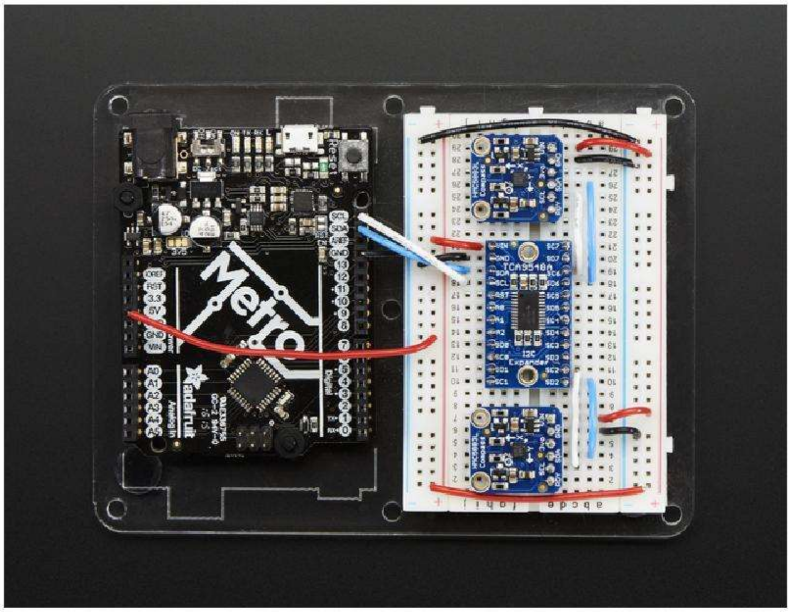
You just found the perfect I2C sensor, and you want to wire up two or three or more of them to your Arduino when you realize "Uh oh, this chip has a fixed I2C address, and from what I know about I2C, you cannot have two devices with the same address on the same SDA/SCL pins!" Are you out of luck? You would be, if you didn't have this ultra-cool **TCA9548A 1-to-8 I2C multiplexer!**



Finally, a way to get up to 8 same-address I2C devices hooked up to one microcontroller - this multiplexer acts as a gatekeeper, shuttling the commands to the selected set of I2C pins with your command.



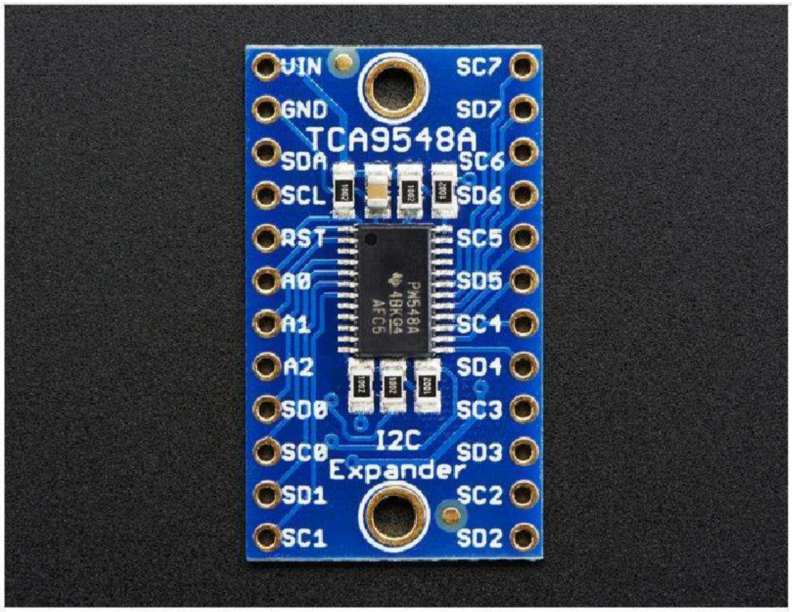
Using it is fairly straight-forward: the multiplexer itself is on I2C address 0x70 (but can be adjusted from 0x70 to 0x77) and you simply write a single byte with the desired multiplexed output number to that port, and bam - any future I2C packets will get sent to that port. In theory, you could have 8 of these multiplexers on each of 0x70-0x77 addresses in order to control 64 of the same-I2C-addressed-part.



Like all Adafruit breakouts, we put this nice chip on a breakout for you so you can use it on a breadboard with capacitors, and pullups and pulldowns to make usage a snap. Some header is required and once soldered in you can plug it into a solderless-breadboard. The chip itself is 1.8V - 5V compliant so you can use it with any logic level.

We even wrote up a nice tutorial with wiring diagrams, schematics and examples to get you running in 10 minutes! (https://adafru.it/jhC)

Pinouts



Power Pins:

- **Vin** - this is the power pin. Since the sensor chip uses 3-5 VDC. To power the board, give it the same power as the logic level of your microcontroller - e.g. for a 5V micro like Arduino, use 5V
- **GND** - common ground for power and logic

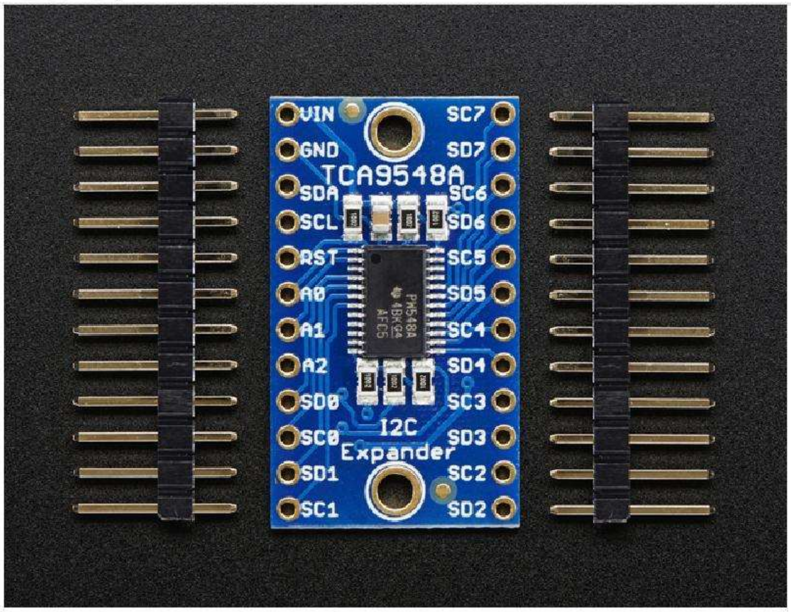
I2C Control-Side pins:

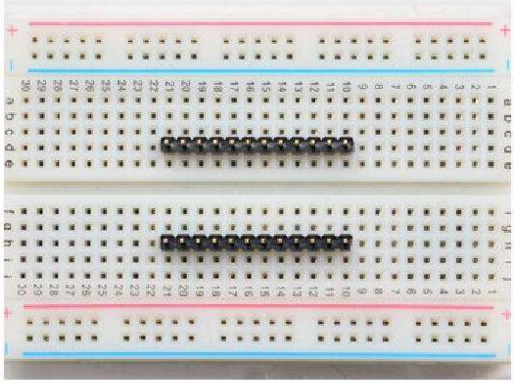
- **SCL** - this is the I2C clock pin for the chip itself, connect to your microcontrollers I2C clock line.
- **SDA** - this is the I2C data pin for the chip itself, connect to your microcontrollers I2C data line.
- **RST** - this is the reset pin, for resetting the multiplexer chip. Pulled high by default, connect to ground to reset
- **A0 A1 A2** - these are the address selection pins *for the multiplexer*. By default the multiplexer is at address **0x70** and these three pins are pulled low. Connect them to **Vin** to set the address to **0x71** - **0x77**.
- **A0** is the lowest-significant bit (if it is pulled high, it will increase the address by 1).
- **A1** is the 2nd-lowest-significant bit (if it is pulled high, it will increase the address by 2).
- **A2** is the 3rd-lowest-significant bit (if it is pulled high, it will increase the address by 4).

I2C Multiplexed-Side pins:

- **SDx** and **SCx**: There are 8 sets of **SDx** and **SCx** pins, from **SD0/SC0** to **SD7/SC7**. These are the multiplexed pins. Each one is a completely seperate I2C bus set. So you have have 8 I2C devices with identical addresses, as long as they are on one I2C bus each.
  These pins do not have any pullups installed, so if you are using a chip or breakout without i2c pullups be sure to add them! Nicely, you can have **Vin** be 3.3V and have these pins pulled up to 5V (that is, they are 5V compliant)

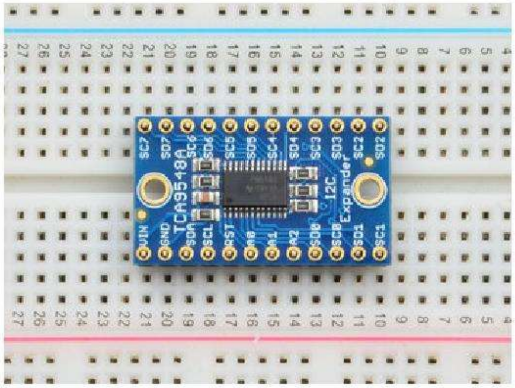
Assembly





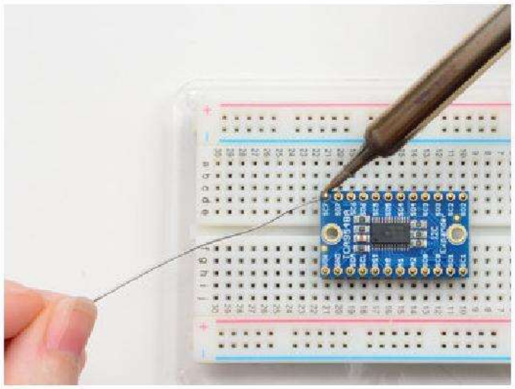
Prepare the header strip:

Cut the strip to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**



Add the breakout board:

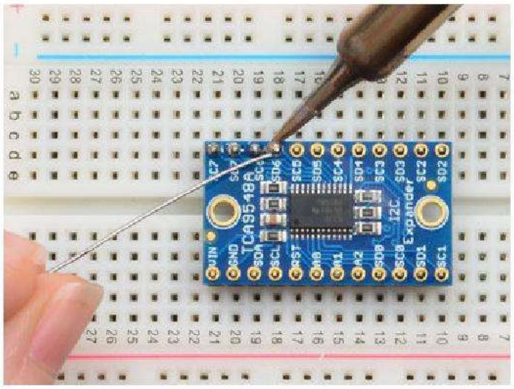
Place the breakout board over the pins so that the short pins poke through the breakout pads

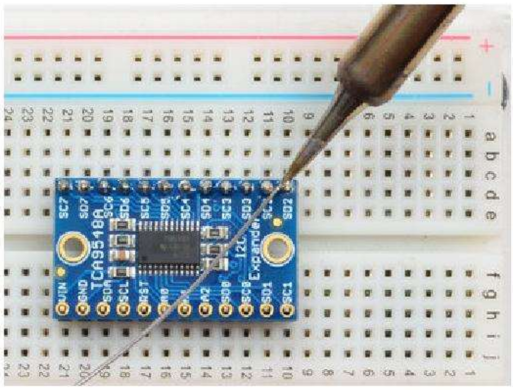


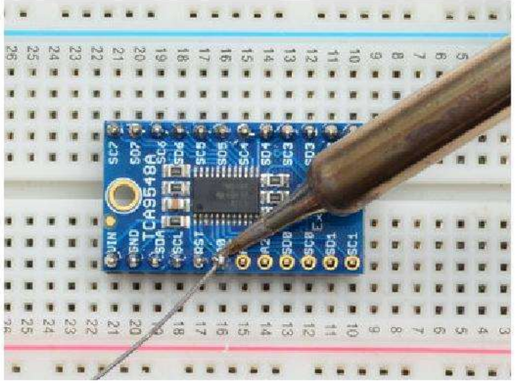
And Solder!

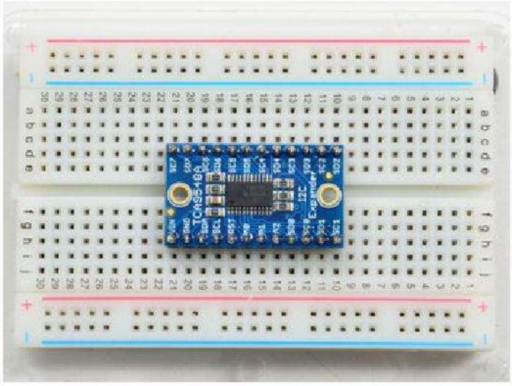
Be sure to solder all pins for reliable electrical contact.

(For tips on soldering, be sure to check out our Guide to Excellent Soldering (https://adafru.it/aTk)*).*


TCA9548A
I2C Expander
VIN GND SDA SCL RST A0 A1 A2 SD0 SC0 SD1 SC1
SC7 SD7 SC6 SD6 SC5 SD5 SC4 SD4 SC3 SD3 SC2 SD2


TCA9548A
I2C Expander


TCA9548A



You're done! Check your solder joints visually and continue onto the next steps

Wiring & Test

The TCA9548A multiplexer is interesting in that it has an I2C address (0x70 by default) - and you basically send it a command to tell it which I2C multiplexed output you want to talk to, then you can address the board you want to address.

We suggest using this little helper to help you select the port

```
#define TCAADDR 0x70

void tcaselect(uint8_t i) {
  if (i > 7) return;

  Wire.beginTransmission(TCAADDR);
  Wire.write(1 << i);
  Wire.endTransmission();
}
```

You can then call tcaselect(0) thru tcaselect(7) to set up the multiplexer.

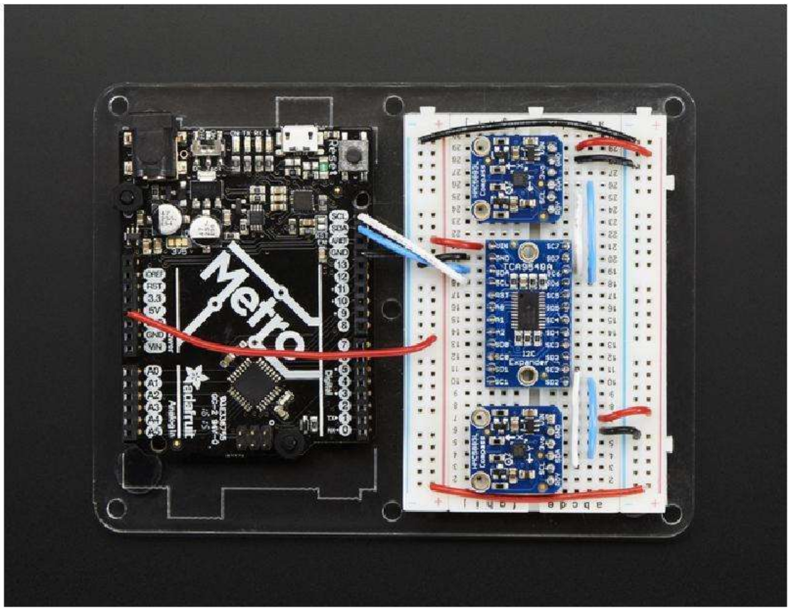
Note that you if you happen to have I2C devices with I2C address 0x70, you will need to short one of the **Addr** pins on the TCA9548 breakout to **Vin** in order to make it not conflict. Given that you can have 0x70 thru 0x77, just find one that's free and you're good to go!

Example Multiplexing

For example, say we want to talk to two HMC5883 breakouts. These magnetometers have a fixed address of **0x1E** so you cannot have two on one I2C bus. Wire up the TCA9548 breakout so that:

- **Vin** is connected to **5V** (on a 3V logic Arduino/microcontroller, use **3.3V**)
- **GND** to ground
- **SCL** to I2C clock
- **SDA** to I2C data

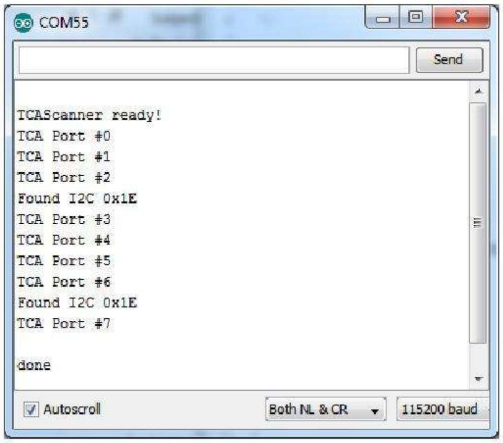
Then wire up each of the other sensor breakouts to **Vin**, **Ground** and use one of the **SC*n*** / **SD*n*** multiplexed buses:



On an Arduino, which is what we're using, we suggest running this handy scanner script which will tell you what the breakout detected

```
/**
 * TCA9548 I2CScanner.pde -- I2C bus scanner for Arduino
 *
 * Based on code c. 2009, Tod E. Kurt, http://todbot.com/blog/
 *
 */

#include "Wire.h"
extern "C" {
#include "utility/twi.h"  // from Wire library, so we can do bus scanning
}

#define TCAADDR 0x70

void tcaselect(uint8_t i) {
  if (i > 7) return;

  Wire.beginTransmission(TCAADDR);
  Wire.write(1 << i);
  Wire.endTransmission();
}


// standard Arduino setup()
void setup()
{
    while (!Serial);
    delay(1000);

    Wire.begin();

    Serial.begin(115200);
    Serial.println("\nTCAScanner ready!");

    for (uint8_t t=0; t<8; t++) {
      tcaselect(t);
      Serial.print("TCA Port #"); Serial.println(t);

      for (uint8_t addr = 0; addr<=127; addr++) {
        if (addr == TCAADDR) continue;

        uint8_t data;
        if (! twi_writeTo(addr, &data, 0, 1, 1)) {
           Serial.print("Found I2C 0x");  Serial.println(addr,HEX);
        }
      }
    }
    Serial.println("\ndone");
}

void loop()
{
}
```

For example, running it on the above setup will give you:



```
TCAScanner ready!
TCA Port #0
TCA Port #1
TCA Port #2
Found I2C 0x1E
TCA Port #3
TCA Port #4
TCA Port #5
TCA Port #6
Found I2C 0x1E
TCA Port #7

done
```

Next up you will have to adjust whatever code you have to select the correct multiplexed port!

Make sure before you query from the sensor that you call **tcaselect** to get the right one

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_HMC5883_U.h>

#define TCAADDR 0x70

/* Assign a unique ID to this sensor at the same time */
Adafruit_HMC5883_Unified mag1 = Adafruit_HMC5883_Unified(1);
Adafruit_HMC5883_Unified mag2 = Adafruit_HMC5883_Unified(2);

void displaySensorDetails(Adafruit_HMC5883_Unified *mag)
{
  sensor_t sensor;
  mag->getSensor(&sensor);
  Serial.println("------------------------------------");
  Serial.print  ("Sensor:       "); Serial.println(sensor.name);
  Serial.print  ("Driver Ver:   "); Serial.println(sensor.version);
  Serial.print  ("Unique ID:    "); Serial.println(sensor.sensor_id);
  Serial.print  ("Max Value:    "); Serial.print(sensor.max_value); Serial.println(" uT");
  Serial.print  ("Min Value:    "); Serial.print(sensor.min_value); Serial.println(" uT");
  Serial.print  ("Resolution:   "); Serial.print(sensor.resolution); Serial.println(" uT");
  Serial.println("------------------------------------");
  Serial.println("");
  delay(500);
}

void tcaselect(uint8_t i) {
  if (i > 7) return;

  Wire.beginTransmission(TCAADDR);
  Wire.write(1 << i);
  Wire.endTransmission();
}
```

```
void setup(void)
{
  Serial.begin(9600);
  Serial.println("HMC5883 Magnetometer Test"); Serial.println("");

  /* Initialise the 1st sensor */
  tcaselect(2);
  if(!mag1.begin())
  {
    /* There was a problem detecting the HMC5883 ... check your connections */
    Serial.println("Ooops, no HMC5883 detected ... Check your wiring!");
    while(1);
  }

  /* Initialise the 2nd sensor */
  tcaselect(6);
  if(!mag2.begin())
  {
    /* There was a problem detecting the HMC5883 ... check your connections */
    Serial.println("Ooops, no HMC5883 detected ... Check your wiring!");
    while(1);
  }

  /* Display some basic information on this sensor */
  tcaselect(2);
  displaySensorDetails(&mag1);
  tcaselect(6);
  displaySensorDetails(&mag2);
}

void loop(void)
{
  /* Get a new sensor event */
  sensors_event_t event;

  tcaselect(2);
  mag1.getEvent(&event);

  /* Display the results (magnetic vector values are in micro-Tesla (uT)) */
  Serial.print("Sensor #1 - ");
  Serial.print("X: "); Serial.print(event.magnetic.x); Serial.print("  ");
  Serial.print("Y: "); Serial.print(event.magnetic.y); Serial.print("  ");
  Serial.print("Z: "); Serial.print(event.magnetic.z); Serial.print("  ");Serial.println("uT");

  tcaselect(6);
  mag2.getEvent(&event);
  /* Display the results (magnetic vector values are in micro-Tesla (uT)) */
  Serial.print("Sensor #2 - ");
  Serial.print("X: "); Serial.print(event.magnetic.x); Serial.print("  ");
  Serial.print("Y: "); Serial.print(event.magnetic.y); Serial.print("  ");
  Serial.print("Z: "); Serial.print(event.magnetic.z); Serial.print("  ");Serial.println("uT");

  delay(500);
}
```

However, once you add all the **tcaselect()**'s you will be able to talk to both sensors!

Multiple Multplexers

Since the TCA9548 is addressible, you can have more than one multiplexer on the bus. With 8 possible adresses, that means you can control as many as 64 separate i2c buses.

To avoid conflict between devices with the same address on different multiplexers, you can disable all channels on a multiplexer with the following code:

```
Wire.beginTransmission(TCAADDR1);
Wire.write(0);  // no channel selected
Wire.endTransmission();
```

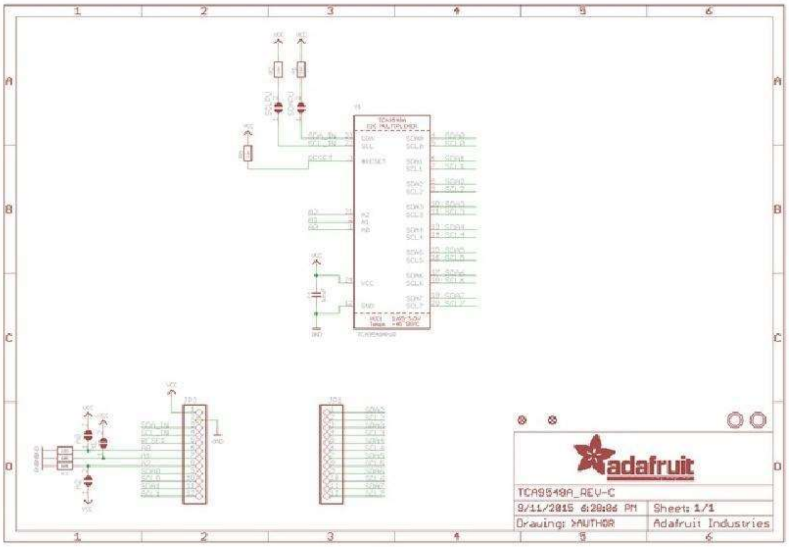
Downloads

Datasheets

- TCA9548A datasheet (https://adafru.it/id8)
- Fritzing object in the Adafruit Fritzing Library (https://adafru.it/aP3)
- EagleCAD PCB files on GitHub (https://adafru.it/rzF)

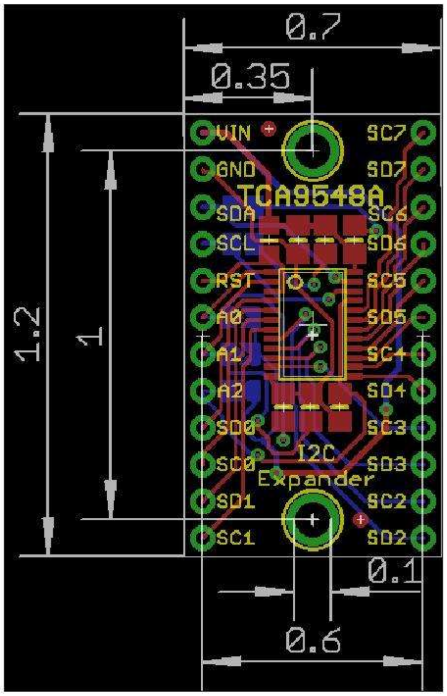
Schematic

Click to embiggen

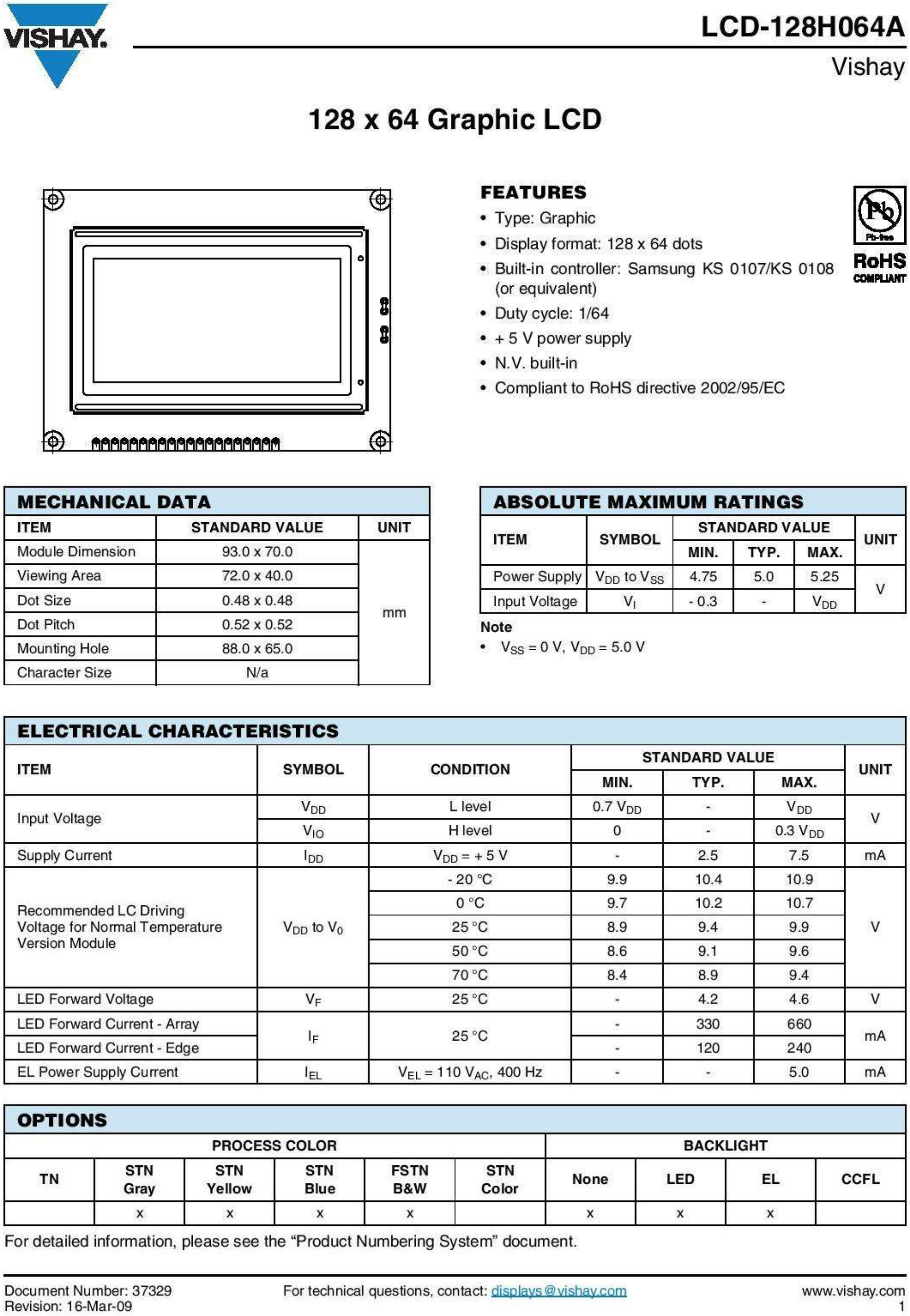


Fabrication Print

Dimensions in Inches


0.7
0.35
1.2
1
VIN
GND
SDA
SCL
RST
A0
A1
A2
SD0
SC0
SD1
SC1
SC7
SD7
SC6
SD6
SC5
SD5
SC4
SD4
SC3
SD3
SC2
SD2
TCA9548A
I2C
Expander
0.1
0.6

10.11 Appendix K: Graphic KS0108 LCD Datasheet



LCD-128H064A
Vishay

128 x 64 Graphic LCD

FEATURES

- Type: Graphic
- Display format: 128 x 64 dots
- Built-in controller: Samsung KS 0107/KS 0108 (or equivalent)
- Duty cycle: 1/64
- + 5 V power supply
- N.V. built-in
- Compliant to RoHS directive 2002/95/EC

Pb-free RoHS COMPLIANT

MECHANICAL DATA

ITEM	STANDARD VALUE	UNIT
Module Dimension	93.0 x 70.0	mm
Viewing Area	72.0 x 40.0	
Dot Size	0.48 x 0.48	
Dot Pitch	0.52 x 0.52	
Mounting Hole	88.0 x 65.0	
Character Size	N/a	

ABSOLUTE MAXIMUM RATINGS

ITEM	SYMBOL	STANDARD VALUE			UNIT
		MIN.	TYP.	MAX.	
Power Supply	V_{DD} to V_{SS}	4.75	5.0	5.25	V
Input Voltage	V_I	- 0.3	-	V_{DD}	

Note

- V_{SS} = 0 V, V_{DD} = 5.0 V

ELECTRICAL CHARACTERISTICS

ITEM	SYMBOL	CONDITION	STANDARD VALUE			UNIT
			MIN.	TYP.	MAX.	
Input Voltage	V_{DD}	L level	0.7 V_{DD}	-	V_{DD}	V
	V_{IO}	H level	0	-	0.3 V_{DD}	
Supply Current	I_{DD}	V_{DD} = + 5 V	-	2.5	7.5	mA
Recommended LC Driving Voltage for Normal Temperature Version Module	V_{DD} to V_0	- 20 °C	9.9	10.4	10.9	V
		0 °C	9.7	10.2	10.7	
		25 °C	8.9	9.4	9.9	
		50 °C	8.6	9.1	9.6	
		70 °C	8.4	8.9	9.4	
LED Forward Voltage	V_F	25 °C	-	4.2	4.6	V
LED Forward Current - Array	I_F	25 °C	-	330	660	mA
LED Forward Current - Edge			-	120	240	
EL Power Supply Current	I_{EL}	V_{EL} = 110 V_{AC}, 400 Hz	-	-	5.0	mA

OPTIONS

PROCESS COLOR						BACKLIGHT			
TN	STN Gray	STN Yellow	STN Blue	FSTN B&W	STN Color	None	LED	EL	CCFL
	x	x	x	x		x	x	x	

For detailed information, please see the "Product Numbering System" document.

Document Number: 37329
Revision: 16-Mar-09

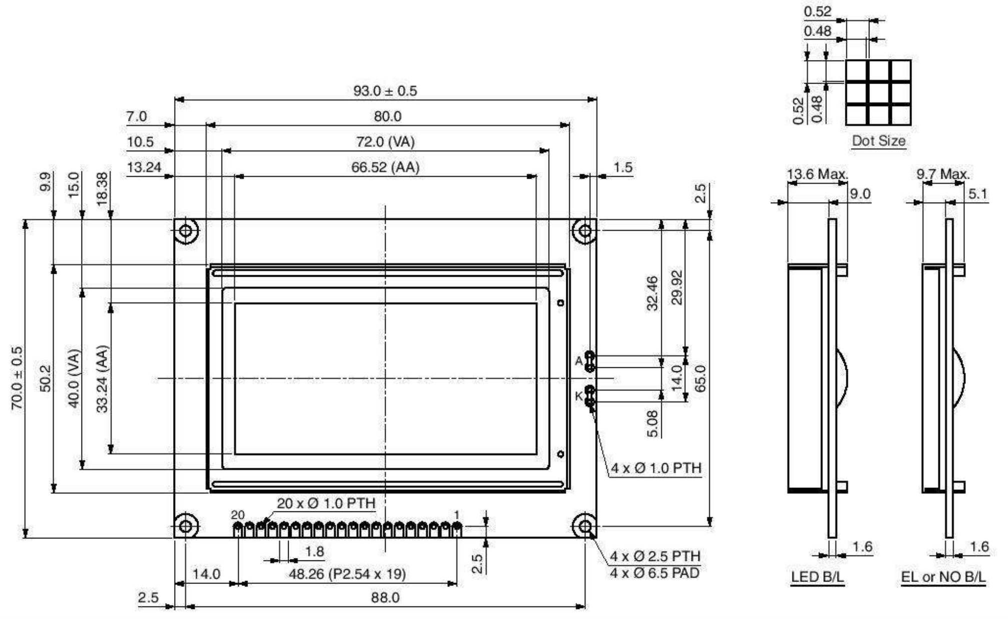
For technical questions, contact: displays@vishay.com

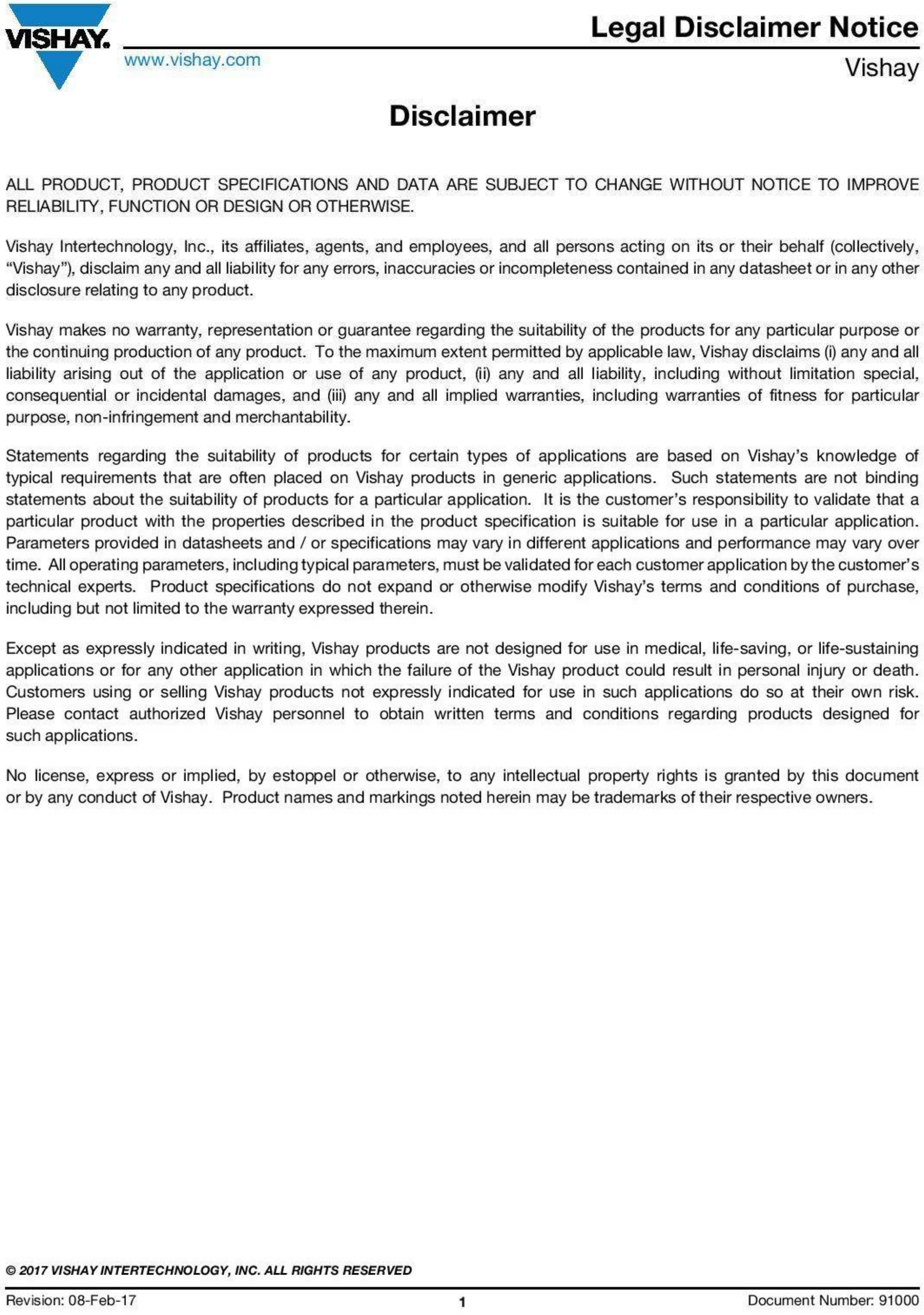
www.vishay.com
1

INTERFACE PIN FUNCTION

PIN NO.	SYMBOL	FUNCTION
1	V_{SS}	Ground
2	V_{DD}	Power supply (+ 5 V)
3	V_0	Contrast adjustment
4	D/I	Data/instruction
5	R/$\overline{W}$	Data read/write
6	E	H → L enable signal
7	DB0	Data bus line
8	DB1	Data bus line
9	DB2	Data bus line
10	DB3	Data bus line
11	DB4	Data bus line
12	DB5	Data bus line
13	DB6	Data bus line
14	DB7	Data bus line
15	CS1	Chip select for IC1
16	CS2	Chip select for IC1
17	$\overline{RST}$	Reset
18	V_{EE}	Negative voltage output
19	A	Power supply for LED (+ 4.2 V), $R_A = 0\ \Omega$
20	K	Power supply for LED (0 V)

DIMENSIONS in millimeters





VISHAY.
www.vishay.com

Legal Disclaimer Notice

Vishay

Disclaimer

ALL PRODUCT, PRODUCT SPECIFICATIONS AND DATA ARE SUBJECT TO CHANGE WITHOUT NOTICE TO IMPROVE RELIABILITY, FUNCTION OR DESIGN OR OTHERWISE.

Vishay Intertechnology, Inc., its affiliates, agents, and employees, and all persons acting on its or their behalf (collectively, “Vishay”), disclaim any and all liability for any errors, inaccuracies or incompleteness contained in any datasheet or in any other disclosure relating to any product.

Vishay makes no warranty, representation or guarantee regarding the suitability of the products for any particular purpose or the continuing production of any product. To the maximum extent permitted by applicable law, Vishay disclaims (i) any and all liability arising out of the application or use of any product, (ii) any and all liability, including without limitation special, consequential or incidental damages, and (iii) any and all implied warranties, including warranties of fitness for particular purpose, non-infringement and merchantability.

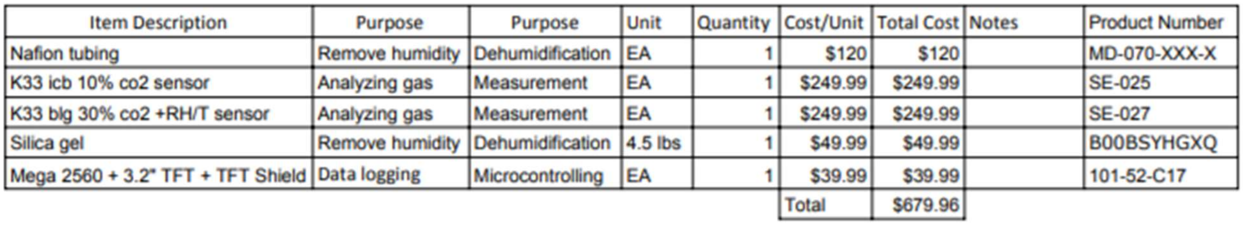
Statements regarding the suitability of products for certain types of applications are based on Vishay’s knowledge of typical requirements that are often placed on Vishay products in generic applications. Such statements are not binding statements about the suitability of products for a particular application. It is the customer’s responsibility to validate that a particular product with the properties described in the product specification is suitable for use in a particular application. Parameters provided in datasheets and / or specifications may vary in different applications and performance may vary over time. All operating parameters, including typical parameters, must be validated for each customer application by the customer’s technical experts. Product specifications do not expand or otherwise modify Vishay’s terms and conditions of purchase, including but not limited to the warranty expressed therein.

Except as expressly indicated in writing, Vishay products are not designed for use in medical, life-saving, or life-sustaining applications or for any other application in which the failure of the Vishay product could result in personal injury or death. Customers using or selling Vishay products not expressly indicated for use in such applications do so at their own risk. Please contact authorized Vishay personnel to obtain written terms and conditions regarding products designed for such applications.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document or by any conduct of Vishay. Product names and markings noted herein may be trademarks of their respective owners.

Revision: 08-Feb-17 1 Document Number: 91000

10.12 Appendix L: Budget



Item Description	Purpose	Purpose	Unit	Quantity	Cost/Unit	Total Cost	Notes	Product Number
Nafion tubing	Remove humidity	Dehumidification	EA	1	$120	$120		MD-070-XXX-X
K33 icb 10% co2 sensor	Analyzing gas	Measurement	EA	1	$249.99	$249.99		SE-025
K33 blg 30% co2 +RH/T sensor	Analyzing gas	Measurement	EA	1	$249.99	$249.99		SE-027
Silica gel	Remove humidity	Dehumidification	4.5 lbs	1	$49.99	$49.99		B00BSYHGXQ
Mega 2560 + 3.2" TFT + TFT Shield	Data logging	Microcontrolling	EA	1	$39.99	$39.99		101-52-C17
					Total	$679.96		

10.13 Appendix M: DHR

DHR

[3D Printing]

MPI Steps	Deviations	Completed by	Signature	Date
1		Tristan Frisella	TF	2/7/19
2		Tristan Frisella	TF	2/7/19
3		Tristan Frisella	TF	2/7/19
4		Tristan Frisella	TF	2/7/19
5		Tristan Frisella	TF	2/8/19
6		Tristan Frisella	TF	2/8/19

[Dehumidifying Chamber]

MPI Steps	Deviations	Completed by	Signature	Date
1	The support was broken prior assembly, this piece was still used but will have to be replaced	Tristan Frisella	TF	2/8/19
2		Tristan Frisella	TF	2/8/19
3		Tristan Frisella	TF	2/8/19
4		Tristan Frisella	TF	2/8/19
5		Tristan Frisella	TF	2/9/19
6		Tristan Frisella	TF	2/9/19
7		Tristan Frisella	TF	2/9/19

[Data Logging Shield]

MPI Steps	Deviations	Completed by	Signature	Date
1		Daniel Alanis	DA	2/9/19

[Screen]

MPI Steps	Deviations	Completed by	Signature	Date
1		Daniel Alanis	DA	2/9/19
2		Daniel Alanis	DA	2/9/19
3		Daniel Alanis	DA	2/9/19
4		Daniel Alanis	DA	2/9/19
5		Daniel Alanis	DA	2/9/19
6		Daniel Alanis	DA	2/9/19
7		Daniel Alanis	DA	2/9/19
8		Daniel Alanis	DA	2/9/19
9		Daniel Alanis	DA	2/9/19
10		Daniel Alanis	DA	2/9/19
11		Daniel Alanis	DA	2/9/19
12		Daniel Alanis	DA	2/9/19
13		Daniel Alanis	DA	2/9/19
14		Daniel Alanis	DA	2/9/19
15		Daniel Alanis	DA	2/9/19
16		Daniel Alanis	DA	2/9/19
17		Daniel Alanis	DA	2/9/19
18		Daniel Alanis	DA	2/9/19
19		Daniel Alanis	DA	2/9/19
20		Daniel Alanis	DA	2/9/19
21		Daniel Alanis	DA	2/9/19
22		Daniel Alanis	DA	2/9/19
23		Daniel Alanis	DA	2/9/19
24		Daniel Alanis	DA	2/9/19
25		Daniel Alanis	DA	2/9/19

[Temperature and Humidity Sensor]

MPI Steps	Deviations	Completed by	Signature	Date
1		Daniel Alanis	DA	2/9/19
2		Daniel Alanis	DA	2/9/19
3	One of the sensors broke, currently reordering new sensor	Daniel Alanis	DA	2/9/19
4		Daniel Alanis	DA	2/9/19
5		Daniel Alanis	DA	2/9/19

[CO_2 Sensor]

MPI Steps	Deviations	Completed by	Signature	Date
1		Daniel Alanis	DA	2/9/19
2		Daniel Alanis	DA	2/9/19
3		Daniel Alanis	DA	2/9/19
4		Daniel Alanis	DA	2/9/19
5		Daniel Alanis	DA	2/9/19
6		Daniel Alanis	DA	2/9/19
7		Daniel Alanis	DA	2/9/19
8		Daniel Alanis	DA	2/9/19
9		Daniel Alanis	DA	2/9/19
10		Daniel Alanis	DA	2/9/19

10.14 Appendix N: Arduino Code

```
////////////////////////////////////////////////////////////////
// Accelerated Wear Dehumidification Chamber Sensors and Data Logging
// California Polytechnic State University, San Luis Obispo - Biomedical Engineering
// Senior Design - Fall 2018/Winter 2019
// Team: Daniel Alanis, Kian Ashoubi, Tristan Frisella
////////////////////////////////////////////////////////////////
#include <openGLCD.h>
#include <Wire.h>
#include "SHTSensor.h"
#include "RTClib.h"
#include <SPI.h>
#include <SD.h>
#define TCAADDR 0x70

////////////////////////////////////// Function to use Multiplexer
void tcaselect(uint8_t i) {
  if (i > 7) return;       // Because there are only 7 locations on the multiplexer, calling tcaselect(#) above 7 will cancel the function

  Wire.beginTransmission(TCAADDR);   // Begin wire transmission to multiplexer
  Wire.write(1 << i);                // Shifts bits to switch which location you're connected to on the multiplexer
  Wire.endTransmission();            // End wire transmission to multiplexer
}
////////////////////////////////////// Error function for SD Card
void error(char *str)
  {
  Serial.print("error: ");
  Serial.println(str);
  }
const int chipSelect = 10;    // chipselect for the shield, therefore it's using pin 10

////////////////////////////////////// Defining I2C address for both CO2 sensors
// 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
// Modified by Accelerated Wear Team - Cal Poly BMED 2019
// We will be using the I2C hardware interface on the Arduino in
// combination with the built-in Wire library to interface.

int co2Addr = 0x66;
int co2Addrb = 0x67;

//////////////////////////////// Initializing T/H sensors via I2C
// T/H Sensor with normal i2c address
// T/H Sensor 1
SHTSensor sht1(SHTSensor::SHT3X);

// T/H Sensor with alternative i2c address
// T/H Sensor 2
SHTSensor sht2(SHTSensor::SHT3X);

////////////////////////////////  Initializing the RTC
// Date and time functions using a DS1307 RTC connected via I2C and Wire lib
RTC_PCF8523 rtc;

char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday"};



void setup() {
GLCD.ClearScreen();
/////////////////////////////////////////////// This is for the real time clock integrated to the SD card
reader

  while (!Serial) {
    delay(1);  // for Leonardo/Micro/Zero
  }

  Serial.begin(9600);                     // opens up serial port for GUI
  if (! rtc.begin()) {
    Serial.println("Couldn't find RTC");
    while (1);
  }

  if (! rtc.initialized()) {
    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));
  }
  ////////////////////////////////////////////////////////// Initialize the GLCD

  GLCD.Init();

  // Select the font for the default text area
  GLCD.SelectFont(System5x7, PIXEL_ON);

  //  GLCD.print(F("hello, world!")); // keep string in flash on AVR boards with IDE 1.x
  //  GLCD.Puts(F("hello, world!")); // Puts() supports F() with any version of IDE

  // print() below uses RAM on AVR boards but works
  // on any version of IDE with any processor
  // note: Same is true for Puts()

  ////////////////////////////////////////////////////////// This is to initialize SD Card

  Serial.print("Initializing SD card...");

  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect)) {                         // initializes pin 11,12,13 on shield to begin data
logging
    Serial.println("Card failed, or not present");
    // don't do anything more:
    while (1);
  }
  Serial.println("card initialized.");
  Serial.begin(9600);
  Wire.begin ();
  Serial.println("Application Note AN-102: Interface Arduino to K-30 and SHT85 T/H");
  // initialize first T/H sensor
  tcaselect(0);   // turns the switch to location 0 on multiplexer
  sht1.init();    // initializes the switch on location 0
  // initialize second T/H sensor
  tcaselect(1);   // turns the switch to location 1 on multiplexer
  sht2.init();    // initializes the switch on location 1
}

////////////////////////////////////////////////////////////////////////////
// Function : int readCO2()
// Returns : CO2 Value upon success, 0 upon checksum failure
// Assumes : - Wire library has been imported successfully.
// - CO2 sensor address is defined in co2_addr
// First function is made for the first CO2 sensor
////////////////////////////////////////////////////////////////////////////
int readCO2()
{
  int co2_value = 0;  // We will store the CO2 value inside this variable.

  //////////////////////////
  /* 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’ 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 necessary. 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!
  return co2_value;
 }
 else
 {
  // Failure!
  /*
    Checksum failure can be due to a number of factors,
    fuzzy electrons, sensor busy, etc.
  */
  return 0;
 }
}
////////////////////////////////////////////////////////////////
// Function : int readCO2b()
// Returns : CO2 Value upon success, 0 upon checksum failure
// Assumes : - Wire library has been imported successfully.
// - CO2 sensor address is defined in co2_addr
// Second function is made for the second CO2 sensor
////////////////////////////////////////////////////////////////
int readCO2b()
{
 int co2_valueb = 0;  // We will store the CO2 value inside this variable.

 //////////////////////////
 /* Begin Write Sequence */
 //////////////////////////
 Wire.beginTransmission(co2Addrb);
 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(co2Addrb, 4);

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

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

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

```
  /*
    Using some bitwise manipulation we will shift our buffer
    into an integer for general consumption
  */
  co2_valueb = 0;
  co2_valueb |= buffer[1] & 0xFF;
  co2_valueb = co2_valueb << 8;
  co2_valueb |= buffer[2] & 0xFF;


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

  if (sum == buffer[3])
  {
    // Success!
    return co2_valueb;
  }
  else
  {
    // Failure!
    /*
      Checksum failure can be due to a number of factors,
      fuzzy electrons, sensor busy, etc.
    */
    return 0;
  }
}

void loop() {
 ////////////////////////////////////////////////////////// This part is to print RTC to GUI
  DateTime now = rtc.now();                    // sets time for clock on shield

  // everything below is printing to GUI with decimal format.
  Serial.print(now.month(), DEC);
  Serial.print('/');
  Serial.print(now.day(), DEC);
  Serial.print('/');
  Serial.print(now.year(), DEC);
  Serial.print(" (");
  Serial.print(daysOfTheWeek[now.dayOfTheWeek()]);      // calls current day of the week
  Serial.print(") ");
  Serial.print(now.hour(), DEC);
  Serial.print(':');
  Serial.print(now.minute(), DEC);
  Serial.print(':');
  Serial.print(now.second(), DEC);
  Serial.println();

////////////////////////////////////// Printing CO2 data to GUI
  tcaselect(2);     // This is where 0x66 gets plugged into (-)
  int co2Value = readCO2();
  if (co2Value > 0)
  {
    Serial.print("CO2 Value: ");
    Serial.println(co2Value);
  }
  else
  {
    Serial.println("Checksum failed / Communication failure");
  }

  tcaselect(3);   // This is where 0x67 gets plugged into (+)
  int co2Valueb = readCO2b();
  if (co2Valueb > 0)
  {
    Serial.print("CO2 Valueb: ");
    Serial.println(co2Valueb);
  }
  else
  {
    Serial.println("Checksum failed / Communication failure with b");
  }

////////////////////////////////////// Printing SHT data to GUI
  // read from first T/H sensor
  tcaselect(0);
  if (sht1.readSample()) {
    Serial.print("SHT1 :\n");
    Serial.print("  RH: ");
    Serial.print(sht1.getHumidity(), 2);
    Serial.print("\n");
    Serial.print("  T:  ");
    Serial.print(sht1.getTemperature(), 2);
    Serial.print("\n");
  } else {
    Serial.print("Sensor 1: Error in readSample()\n");
  }

  // read from second T/H sensor
  tcaselect(1);
  if (sht2.readSample()) {
    Serial.print("SHT2:\n");
    Serial.print("  RH: ");
    Serial.print(sht2.getHumidity(), 2);
    Serial.print("\n");
    Serial.print("  T:  ");
    Serial.print(sht2.getTemperature(), 2);
    Serial.print("\n");
  } else {
    Serial.print("Sensor 2: Error in readSample()\n");
  }
  delay(1000);

  ///////////////////////////////////////////////////////// Data logging to the SD card

  // make a string for assembling the data to log:
  //Data logging co2(-) data
  String dataStringco2 = "";
  dataStringco2 += String(co2Value);

  char co2minus[] = "co2minusdata.csv";
  File dataFile_minus = SD.open("co2-data.csv", FILE_WRITE);   //Identifies filepath and mode.
FILE_WRITE opens file for reading and writing

  if (! dataFile_minus) {
  error("couldnt create file");
  }
  if (dataFile_minus) {
    dataFile_minus.print(dataStringco2);
    dataFile_minus.print(",");
    dataFile_minus.print(now.month(), DEC);
    dataFile_minus.print('/');
    dataFile_minus.print(now.day(), DEC);
    dataFile_minus.print('/');
    dataFile_minus.print(now.year(), DEC);
    dataFile_minus.print(" (");
    dataFile_minus.print(daysOfTheWeek[now.dayOfTheWeek()]);      // calls current day of the
week
    dataFile_minus.print(") ");
    dataFile_minus.print(now.hour(), DEC);
    dataFile_minus.print(':');
    dataFile_minus.print(now.minute(), DEC);
    dataFile_minus.print(':');
    dataFile_minus.print(now.second(), DEC);
    dataFile_minus.println();
    dataFile_minus.close();
  }

  // if the file isn't open, pop up an error:
  else {
    Serial.println("error opening co2-data.csv");
  }
  //Data logging co2(+) data
  String dataString = "";
  dataString += String(co2Valueb);

  char fileName[] = "co2data.csv";
  File dataFile = SD.open("co2data.csv", FILE_WRITE);   //Identifies filepath and mode.
FILE_WRITE opens file for reading and writing

  if (! dataFile) {
  error("couldnt create co2data.csv file");
  }

  if (dataFile) {
    dataFile.print(co2Valueb);
    dataFile.print(",");
    dataFile.print(now.month(), DEC);
    dataFile.print('/');
    dataFile.print(now.day(), DEC);
    dataFile.print('/');
    dataFile.print(now.year(), DEC);
    dataFile.print(" (");
    dataFile.print(daysOfTheWeek[now.dayOfTheWeek()]);      // calls current day of the week
    dataFile.print(") ");
    dataFile.print(now.hour(), DEC);
    dataFile.print(':');
    dataFile.print(now.minute(), DEC);
    dataFile.print(':');
    dataFile.print(now.second(), DEC);
    dataFile.println();
    dataFile.close();
  }

  // if the file isn't open, pop up an error:
  else {
    Serial.println("error opening co2data.csv");
  }

  ///// Temp and Humidity data for SHT1
  String dataStringSH1 = "";
  dataStringSH1 += String(sht1.getHumidity(), 2);

  String dataStringST1 = "";
  dataStringST1 += String(sht1.getTemperature(), 2);

  char filenameTH1[] = "THdata1.csv";
  File dataFileTH1 = SD.open("TH1data.csv", FILE_WRITE);   //Identifies filepath and mode.
FILE_WRITE opens file for reading and writing

  if (! dataFileTH1) {
  error("couldnt TH1data.csv create file");
  }

  if (dataFileTH1) {
    dataFileTH1.print(dataStringSH1);
    dataFileTH1.print('%');
    dataFileTH1.print(",");
    dataFileTH1.print(dataStringST1);
    dataFileTH1.print('C');
    dataFileTH1.print(",");
    dataFileTH1.print(now.month(), DEC);
    dataFileTH1.print('/');
    dataFileTH1.print(now.day(), DEC);
    dataFileTH1.print('/');
    dataFileTH1.print(now.year(), DEC);
    dataFileTH1.print(" (");
    dataFileTH1.print(daysOfTheWeek[now.dayOfTheWeek()]);      // calls current day of the
week
    dataFileTH1.print(") ");
    dataFileTH1.print(now.hour(), DEC);
    dataFileTH1.print(':');
    dataFileTH1.print(now.minute(), DEC);
    dataFileTH1.print(':');
    dataFileTH1.print(now.second(), DEC);
    dataFileTH1.println();
    dataFileTH1.close();
  }

  // if the file isn't open, pop up an error:
  else {
    Serial.println("error opening TH1data.csv");
  }

  ///// Temp and Humidity data for SHT2
  String dataStringSH2 = "";
  dataStringSH2 += String(sht2.getHumidity(), 2);

  String dataStringST2 = "";
  dataStringST2 += String(sht2.getTemperature(), 2);

  char filenameTH2[] = "THdata2.csv";
  File dataFileTH2 = SD.open("TH2data.csv", FILE_WRITE);   //Identifies filepath and mode.
FILE_WRITE opens file for reading and writing

  if (! dataFileTH2) {
  error("couldnt create TH2data.csv file");
```

```
 }

 if (dataFileTH2) {
   dataFileTH2.print(dataStringSH2);
   dataFileTH2.print('%');
   dataFileTH2.print(",");
   dataFileTH2.print(dataStringST2);
   dataFileTH2.print('C');
   dataFileTH2.print(",");
   dataFileTH2.print(now.month(), DEC);
   dataFileTH2.print('/');
   dataFileTH2.print(now.day(), DEC);
   dataFileTH2.print('/');
   dataFileTH2.print(now.year(), DEC);
   dataFileTH2.print(" (");
   dataFileTH2.print(daysOfTheWeek[now.dayOfTheWeek()]);      // calls current day of the
week
   dataFileTH2.print(") ");
   dataFileTH2.print(now.hour(), DEC);
   dataFileTH2.print(':');
   dataFileTH2.print(now.minute(), DEC);
   dataFileTH2.print(':');
   dataFileTH2.print(now.second(), DEC);
   dataFileTH2.println();
   dataFileTH2.close();
 }

 // if the file isn't open, pop up an error:
 else {
   Serial.println("error opening TH2data.csv");
 }
 /////////////////////////////////////////////////////////// attempting to print RTC to LCD

 GLCD.print(now.month(), DEC);
 GLCD.print('/');
 GLCD.print(now.day(), DEC);
 GLCD.print('/');
 GLCD.print(now.year(), DEC);
 GLCD.print(',');
 GLCD.print(now.hour(), DEC);
```

```
  GLCD.print(':');
  GLCD.print(now.minute(), DEC);
  GLCD.print(':');
  GLCD.print(now.second(), DEC);
  //GLCD.println();

////////////////////////////////////////////////////////// printing the temp & humidity readings to the LCD
  GLCD.print("\nRH1:");   // the '\n' is to print to the next line on the screen
  GLCD.print(sht1.getHumidity(), 2);
  GLCD.print("%");
  GLCD.print("  ");
  GLCD.print("T1:");
  GLCD.print(sht1.getTemperature(), 2);
  GLCD.print("C");
  //second SHT sensor
  GLCD.print("\nRH2:");
  GLCD.print(sht2.getHumidity(), 2);
  GLCD.print("%");
  GLCD.print("  ");
  GLCD.print("T2:");
  GLCD.print(sht2.getTemperature(), 2);
  GLCD.print("C");
  GLCD.println();
////////////////////////////////////////////////////////// printing the co2 readings to the LCD
  GLCD.print("Co2(-) ppm = ");
  GLCD.print(co2Value); //print value
  GLCD.print("\nCo2(+) ppm = ");
  GLCD.print(co2Valueb); //print value
  GLCD.println();
  GLCD.println();
  GLCD.println();
  GLCD.println();
  //delay(1000); //wait 1 second
  //GLCD.ClearScreen();
}
```