SENIOR DESIGN 2

ECEN 4024

Oklahoma State University

Air Flow Sensor & Data Logger

Instructor: *Dr. Krazinski*

Project Advisor: *Dr. Bach*

Group 1A

Report Prepared By:

Stacy Dalzell

Kevin Hill

Aaisha Khalid

Connor McCurley

Table of Contents

Table Of Figures, Tables & Graphs

ECEN 4024 Airflow Sensor Team 1A Final Project Report

Stacy Dalzell, Kevin Hill, Aaisha Khalid, and Connor McCurley

Abstract *(Connor McCurley)* — The problem of heating and cooling buildings efficiently continues to present a significant concern for facility managers throughout the world. Factors contributing to air leakage, such as opening a door or general escape through a building's crevices, greatly decrease the system's efficiency, which in turn, translates to higher operating costs. By measuring the temperature and flow of air into a room and comparing it to the rate of return, the air conditioning system's efficiency can be determined and areas of inefficiency can be resolved. Accordingly, there is a need for monitoring equipment which can record airflow characteristics in an isolated section of ventilation duct for extended periods of time. The objective for this research project was to design and implement a prototype for a low-power, low-restriction airflow and temperature logger which is capable of autonomous operation for at least one month. This report, which serves as a methodological description for the design and implementation of our chosen sensor system, demonstrates a proof of concept for a device which is capable of autonomously measuring the temperature and rate of airflow through a ventilation duct for a period of over two months. Our system, which utilizes a low-profile 3.3V battery to supply an ultra-low power processor, employs custom-fabricated PCB boards containing analog temperature and IR distance sensors to take measurements from a vane flapper. Our prototyped design surpasses specifications in the areas of power regulation, temperature reading, response time, and flow allowance. However, it falls short in the operations of data logging and reliable airflow reading accuracy. With further development, our sensor system has the potential to prove a viable option for efficiency monitoring in large-scale deployments.

1 INTRODUCTION *(KEVIN HILL)*

FOR ECEN 4024 or the Capstone Design course at Oklahoma State University, students are tasked with completing a unique and open-ended electrical engineering project using the skills and information they have garnered over the course of their educational careers. The design group 1A is comprised of four senior electrical engineering students with specializations in power, communication and control systems, as well as computers and programming. This group is tasked with the development of an autonomous, low cost, temperature and airflow sensor and data logger that will be deployed throughout the ventilation system of a medium to large building in a batch of 50 to 100 units. Once in the ventilation system, these devices will autonomously log airflow and temperature data which will be gathered and analyzed by the end users and will be used to measure the heating and ventilation efficiency of the building. With this information, improvements to the building's efficiency can be made, which will reduce energy consumption. This report examines the group's development process and details the final design and implementation results.

2 BACKGROUND *(AAISHA KHALID)*

HVAC duct systems provide thermal comfort, ventilation and acceptable air quality to occupied spaces within buildings. Ducts supply the conditioned air out to the occupied spaces. HVAC systems use air filters, supply fans, return air systems and cooling/heating controls to adjust the air quality and temperature of occupied spaces. Airflow sensors, thermostats, and manometers are often used to test for adequate temperature and air quality.
Maintenance of HVAC systems usually include periodically changing out filters, testing cooling/heating controls, calibrating thermostats and airflow sensors, and replacing/recharging HVAC system power supplies.
This project seeks to improve the testing methods commonly used in HVAC systems. The project scope includes testing accuracy of airflow and temperature sensors, optimizing power source of the system, and reducing cost and time by minimizing maintenance requirements.

3 Design Requirements *(Kevin Hill)*

Accurate measurements require the device to cause minimal airflow obstruction when placed in a straight section of intact four-inch ductwork beyond the first bend after the register opening. A battery capable of providing a minimum of thirty consecutive days of uninterrupted recording is required to fulfill the objective.

The device must additionally tolerate established parameters of temperature variance within the duct, ranging between 4° and 55° C.

The device should be user-friendly to enable the average maintenance individual ease of use to install, activate, and maintain the device as well as recover data.

Specifications	Description
Airflow Accuracy	Within 10% from 150-1500 FPM
Airflow Accuracy	Within 15 FPM for < 150 FPM
Temperature Accuracy	Within 0.5ºC at 0ºC
Operating Temperature	4-55ºC
Battery Life	>30 Days

Table 3.1: Project Specifications

4 Methodology *(Stacy Dalzell)*

The design process was divided into four phases - research, design definition, prototype, and final design. Each of these phases included individual process steps.

4.1 Research Phase

The research phase was used to develop background information, identify state of the art techniques, and acquire information to develop design goals and constraints. Several methods of research were used, including but not limited to searching online journal databases, websites, printed material, and manufacturer datasheets and application notes.

4.2 Design Definition Phase

During design definition phase, the background information from the research phase was used to develop a refined problem statement. From this refined problem statement, a set of design goals and constraints were written. These design goals and constraints were revisited several times for revision during the prototype phase.

4.3 Prototype Phase

This phase was divided into several smaller processes to develop and test design options. For each sub-system, research was performed to identify and rank designs on how they could potentially satisfy the design goals while not violating the constraints. The top design options were evaluated and a prototype design was selected.

Research was performed to identify suitable components for the sub-system design. These components were acquired and the sub-system was assembled using breadboard prototyping methods for testing. Each system was tested to verify datasheet characterization data and system performance. If the system design or component failed to meet the design goals or constraints, research was performed to identify a replacement component or a new design was developed.

Standard experimental methods were used during testing and oscilloscopes, multi-meters, power supplies, and other lab equipment were used to collect data. Some parameters required test fixtures to be constructed or specialized equipment to be acquired. These fixtures and equipment will be discussed in the testing methods section below.

At the conclusion of the prototype phase a set of project specifications were written. The project specifications would be used to determine if design goals were met. A copy of the project specifications is in Appendix E.

4.4 Final Design Phase

During the final design phase, the prototype results were reviewed and designs were finalized. The final design phase included the capture of circuit schematics, verification of interfaces, and mechanical component design and fabrication.

Several software tools were utilized for the final design. SolidWorks 3D modeling software was used to create the designs for the mechanical parts. These parts were created using a 3D printer. The schematic capture and PCB design was accomplished using Altium Designer. Software development was accomplished using the Keil toolchain for the selected processor.

The final design was constructed and assembled for testing. Each sub-system was evaluated to determine if project specifications were met. The entire design was tested for functionality and compliance with design goals and constraints.

5 Design

This section contains a breakdown of individual subsystems. The specifications for each subsystem as well as design decisions to meet these requirements are discussed in detail.

5.1 Airflow Sensor *(Connor McCurley)*

The airflow sensor sub-system was given requirements to meet the following:

- Response Time: < 30 seconds.
- Accuracy: 10% between 150-1500 FPM, ±15 FPM below 150FPM.
- Require no ductwork alterations.
- Provide minimal airflow restriction.

Nine alternatives were initially considered for the airflow sensor.

Airflow Sensor Alternatives
Vane Flapper/Distance
Ultrasonic-Time of Flight
Laser-Doppler Shift
Anemometer
Hotwire
MEMS Pressure
Spring Loaded Vane
Orifice Plate
Barometric Pressure

Table 5.1.1: Airflow Sensor Alternatives

These were narrowed down to two alternatives, each of which had the potential to meet system specifications: the ultrasonic time of flight sensor and the vane flapper with a distance sensor. The comparison between the two is shown in Table 5.1.2.

Sensor System	Pros	Cons
Vane Flapper	• 2.8V input • Easy voltage step-down • Simple implementation • Can be deployed on custom PCB • Low unit cost • Small current draw	• Susceptible to turbulence • Moving components • Potentially less accurate
Ultrasonic	• Potentially improved accuracy • No moving components • Self-contained	• 5V input • Requires voltage step-up • Expensive • Cannot implement on custom board

Table 5.1.2: Vane Flapper and Ultrasonic Pros & Cons

The airflow sub-system designer ultimately selected the vane-flapper with a distance sensor for its ease of implementation and 2.8V voltage requirement, which easily interfaced with the chosen processor board, the Cortex M0.

The VL6180X was selected as the IR distance sensor to read the position of the vane as it was displaced by airflow. The datasheet for the VL6180X provided the specifications below [1]:

- Ranging Accuracy: Accurate to the millimeter between 0-100mm.
- Response Time: < 1.2 ms.
- Standby Current Consumption: < 1µA.
- Ranging Current Consumption: < 1.7mA.
- I2C data communication.

The circuit diagram for the VL6180X is shown in figure 5.1.1 and the actual sensor is shown in figure 5.1.2.

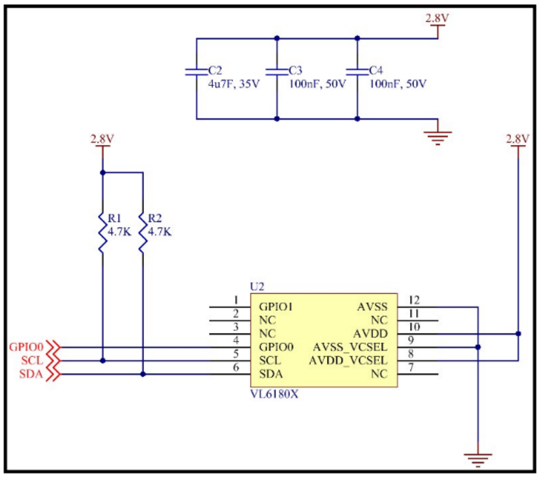


Figure 5.1.1: VL6180X Circuit Diagram

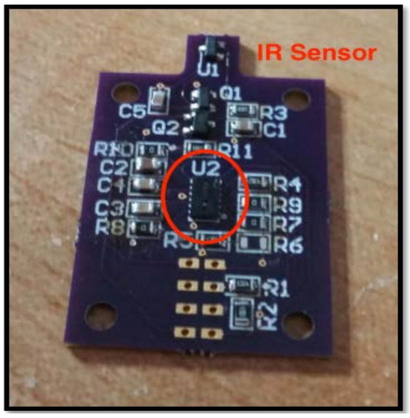


Figure 5.1.2: VL6180X on custom fabrication

The VL6180X interfaces to the Cortex M0 through the utilization of a 2.8V input pin, ground pin, enable pin, SDA pin, and SCL clock pin.

5.2 Temperature Sensor *(Stacy Dalzell)*

The temperature sensor sub-system had to meet the following design requirements:

- Accuracy: ±0.5° C @ 0° C.
- Temperature Range: 4° - 55° C.
- Response Time: Less than 30 seconds.
- Ultra-low power design.

5.2.1 Temperature sensor design

Several methods of measuring temperature were explored and an analog CMOS temperature sensor was selected. The datasheet for the Microchip TC1047 provided the specifications below [2]:

- Temperature Range: -40° - 125° C.
- Typical Temperature Accuracy: ±0.5° C.
- Typical Supply Current: 35 µA.
- Linear output voltage response (10 mV/°C).

The datasheet did not provide sensor response time specifications; however, during testing, the response time was determined to be acceptable for this design. Testing of this sensor will be detailed in the Testing section below.

5.2.2 Circuit Design

The circuit for the TC1047 is shown in the figure below.

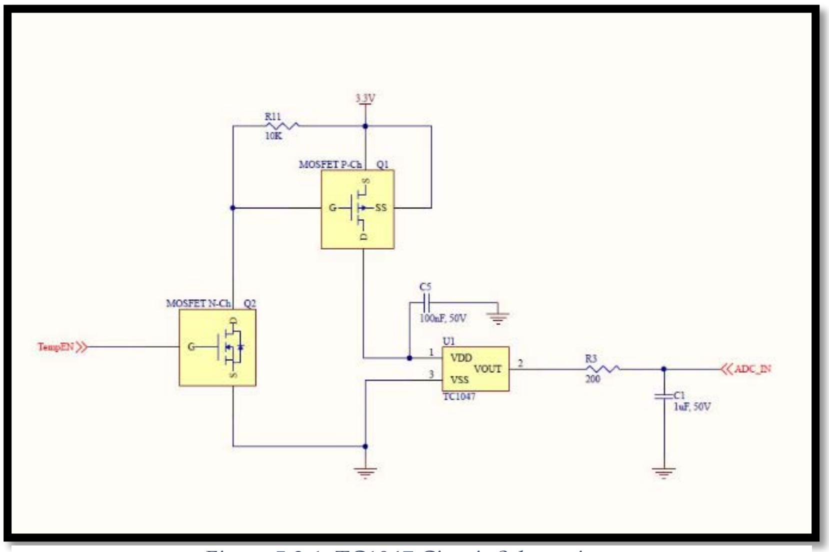


Figure 5.2.1: TC1047 Circuit Schematic

This circuit incorporates the temperature sensor, a filter circuit, and a power conservation circuit. The sensor V_{DD} pin was connected to the 3.3V rail via the power conservation circuit. The V_{SS} pin was connected to the ground rail ant he temperature sensor output pin was connected to a passive low pass filter input.

The power conservation circuit was incorporated to conserve temperature sensor power during device sleep state. This circuit consisted of a PMOS transistor and a NMOS transistor. The PMOS transistor provided the connection for the TC1047 to the 3.3V rail. The gate of the PMOS transistor was pulled up to 3.3V with a 10K resistor. The drain of the NMOS transistor was connected to the gate of the PMOS transistor. The source of the NMOS transistor was connected to ground and the gate was controlled by the microcontroller. When the gate of the NMOS transistor is low, the gate of the PMOS transistor is pulled high. This disconnects the temperature sensor from the 3.3V rail and reduces power consumption. When a temperature measurement is needed, the microcontroller outputs a high logic signal to the gate of the NMOS transistor, providing a ground for the gate of the PMOS transistor. This results in the TC1047 being connected to the 3.3V rail.

A low pass filter was used to provide a stable signal to the ADC of the microcontroller. A passive filter was selected to conserve power and to minimize the size of the circuit. This low pass filter was constructed with a cut off frequency of 795.8 Hz to pass DC voltage to the ADC from the temperature sensor.

5.3 Power *(Aaisha Khalid)*

The power specifications for the airflow sensor and logger project required battery life to exceed 30 days. The power systems engineer determined that several performance parameters needed to be met in selecting an efficient battery. The chosen battery needed to have a long shelf life, high energy density, low self-discharge, minimum memory problems, and minimal impact on the environment.

Four different types of batteries were researched to use for the project, including Nickel Metal Hydride (NiMH), Nickel Cadmium (NiCad), Alkaline, and Lithium polymer batteries. Pros and cons of each battery type were analyzed and the battery that was most compatible with project specifications was chosen as the power source. Table 5.3.1 below summarizes the pros and cons of each battery type.

POWER	PROS	CONS
NiMH	• Energy density • More Environmentally friendly • Average purchase cost • Availability • Simple charging	• Limited discharge current • Limited service life • High self discharge
NiCad	• Fast and simple charge • Long shelf life • Good low temp perf.	• Low energy density charge • Charge memory • Environmentally Unfriendly • Availability
Alkaline	• Availability • Purchase cost	• One time use • Environmentally unfriendly • Cost over time • Low energy density
Lithium Polymer	• High energy density • Light weight • Low self discharge • No memory issues	• Charging (Note 7/ Boeing 787) problems • Transportation regulations • Environmentally unfriendly

Table 5.3.1: Pros & Cons of battery

The power systems engineer also compared battery efficiency, availability, cost, customer reviews, and estimated shipping time in selecting the final battery for the project. Ultimately a lithium polymer/ion (ROHS 08483) battery was selected as the best power source for the airflow sensor.

The battery shown in the picture below was used as the power source for the airflow sensor. This specific lithium polymer battery has many impressive features, including availability in various power ranges, and dimensions. It also has a built-in over-current/voltage protection circuit. Multiple portable chargers are also available for this battery and per unit, the cost of this battery is relatively reasonable. Availability and early shipping of the battery were other driving factors in the decision to use this battery.

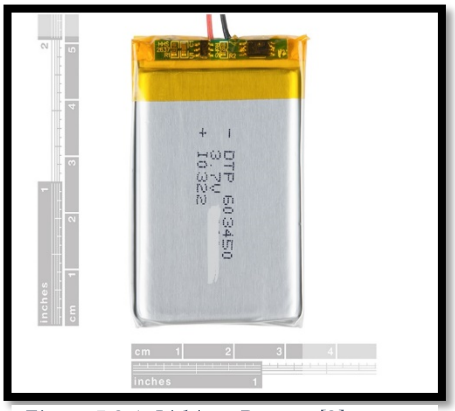


Figure 5.3.1: Lithium Battery [3]

Table 5.3.2 below shows some specifications of the lithium battery.

	Lithium Battery	
Description	**Numerical Values**	**Units**
Nominal Voltage	3.7	Volts
Load Current	1 – 2	C-rate
Operating Temperature	-20 – 60	Celsius
Charge Time	2-4	hours
Typical Capacity	2000	mAh
Maintenance Requirement	None	
Typical Battery Cost	0.14	USD cost per cycle
Battery Dimensions	5.8 x 54 x 60	mm
Battery Weight	36	Grams per battery

Table 5.3.2: Battery Specifications

A buck-boost regulator was selected to minimize power consumption and increase/decrease the available voltage from the battery. Buck-boost regulators can step up and step down voltage. Table 5.3.3 below displays the pros and cons of linear and switching regulators.

POWER – VOLTAGE REGULATOR	PROS	CONS
LINEAR	• Easier to implement • Cost	• Efficiency (27%) • Heat
SWITCHING	• Efficiency (> 90%) • Low heat • Buck, Boost, Buck-Boost	• Cost • Noise • Complexity • Size

Table 5.3.3: Pros & cons Regulators

The battery nominal voltage of 3.7V needed to be stepped down to match the other systems of the airflow sensor. After testing multiple regulators, the TPS61130 buck-boost regulator was determined to be the best regulator for the project's power system. The TPS61130 regulator had many qualities that improved the power system design, but initially the dynamic input voltage range and the regulator efficiency were the most attractive features of this regulator. Figure 5.3.2 below shows an overview of the circuit design for the voltage regulator [3].

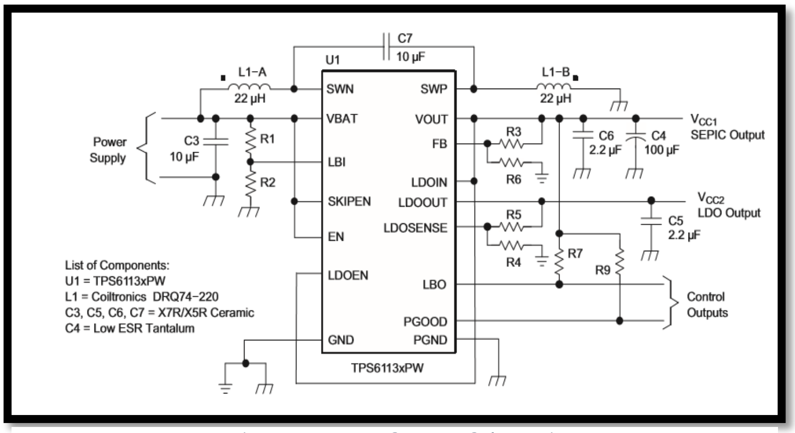


Figure 5.3.2: TPS61130 Schematic

Corresponding datasheets for the regulator provided equations to calculate values for many of the components. The component values were recalculated as the output voltage requirements changed. Figure 5.3.3 below displays equations provided in the datasheet [3].

$$R5 = R4 \times \left(\frac{V_O}{V_{FB}} - 1\right) = 180\ k\Omega \times \left(\frac{V_O}{500\ mV} - 1\right)$$

$$R1 = R2 \times \left(\frac{V_{BAT}}{V_{LBI-threshold}} - 1\right) = 390\ k\Omega \times \left(\frac{V_{BAT}}{500\ mV} - 1\right)$$

$$R3 = R6 \times \left(\frac{V_O}{V_{FB}} - 1\right) = 180\ k\Omega \times \left(\frac{V_O}{500\ mV} - 1\right)$$

$$L1-A = L1-B = \frac{V_{BAT} \times V_{OUT}}{\Delta I_L \times f \times (V_{OUT} + V_{BAT})}$$

Figure 5.3.3: Circuit equations TPS61130

5.4 Processor *(Kevin Hill)*

This design required a processor which supports the peripherals necessary for communicating with the sensors and the SD card. An ADC is necessary in order to read an analog voltage coming from the temperature sensor, the TC1047. Next, the processor must support the inter-integrated circuit or I2C interface to receive the airflow information from the VL6180X distance sensor. Lastly, in order to write data to a micro-SD card, the processor must support the serial peripheral interface or SPI. While supporting these three interfaces is the most important part of the processor choice, to conserve power, the processor should also support some sort of sleep mode. In order to do this, the chosen processor should have predefined sleep modes and a real-time clock or RTC to keep track of when to wake up. After reviewing multiple processors, including those from Atmega, PIC, and others, the processor chosen for this design is the ARM Cortex M0. While many of the other processors supported the three needed peripherals, the Cortex M0 does so while also supporting sleep functions and a RTC which can be driven from either an external or internal oscillator. These sleep functions are used in order to decrease the power consumption by the processor.

5.5 Enclosure

5.5.1 Processor Enclosure (Aaisha Khalid)

The maximum processor/battery enclosure size specifications were 3 x 4 x 1.5″ (L x W x H). The battery's physical features were rectangular, as were the features of the processor board. The physical similarities of both systems were utilized to design one enclosure to house the battery and the processor board.

Ease of access was considered in the design of the processor/battery enclosure as it housed both the battery and the processor board SD card, which were designed for periodic removal to accommodate airflow sensor system maintenance. The designed enclosure was detachable into top and bottom pieces and bound together with screws. The physical dimensions of the bound detachable enclosures were 2.5 x 3.5 x 1.23″ (L x W x H). The top enclosure piece dimensions were 2.5 x 3.5 x 0.6″ (L x W x H). The bottom enclosure piece dimensions were 2.5 x 3.5 x 0.63 (L x W x H). An opening within the enclosure was also required for an RJ45 connector. A side view of the enclosure is shown below.

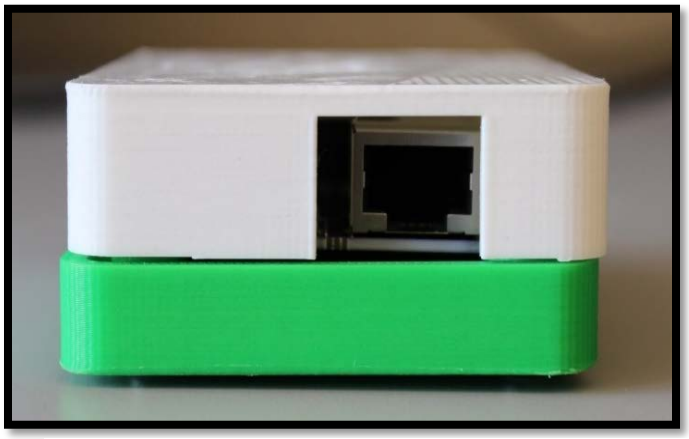
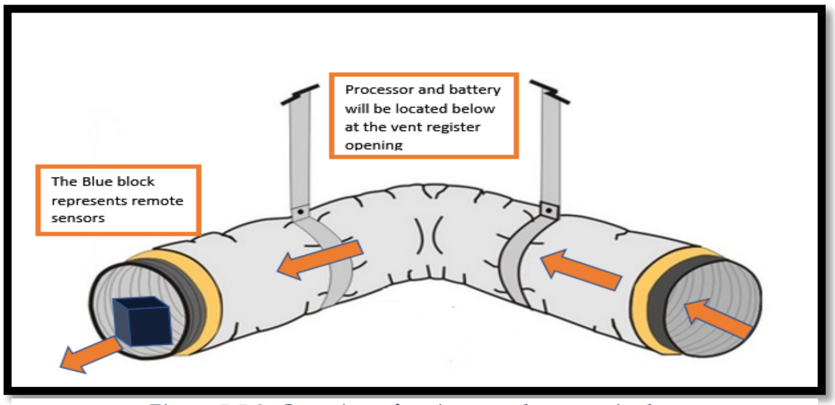


Figure 5.5.1 : Side view (RJ45 connector) of enclosure

Ease of access being a necessity for the enclosure, it is recommended to place the enclosure at the vent register opening of the duct, as seen in the overview drawing shown below [3].


Processor and battery will be located below at the vent register opening
The Blue block represents remote sensors
Figure 5.5.2: Overview of equipment placement in duct

5.5.2 Sensor Enclosure (Connor McCurley)

The sole requirements for the airflow sensor housing were to 1) provide minimal restriction of airflow and 2) require no alteration to the ductwork for installation. To provide proper functionality, the housing needed to hold a vane flapper and allow at least 90 degrees of rotation. It also required the capability to hold the VL6180X at an angle which provided the maximum distance variation as the flapper rotated from 0 to 90 degrees. It was determined that a 45-degree angle, placed 1 inch behind the flapper would provide the optimal reading range for distance sensing. The first iteration of the airflow sensor's housing is shown in the figure below.

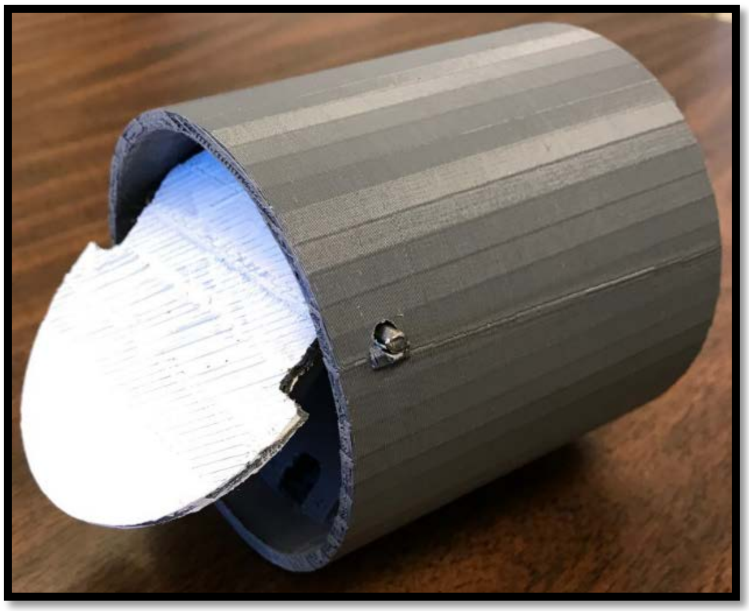


Figure 5.5.3: First generation airflow sensor housing

It was determined that the length of this design was too great to clear the unit around the first bend in the ventilation duct. The second and final design for the airflow sensor's housing is shown in figures 5.5.4, 5.5.5, and 5.5.6. This housing, 3D printed from PLA plastic, uses a minimal amount of material. Its diameter matches the 4" air duct, which provides a flush fit inside the duct, hence meeting the airflow restriction requirements. The VL6180X easily fits onto pegs to ensure its security during operation.

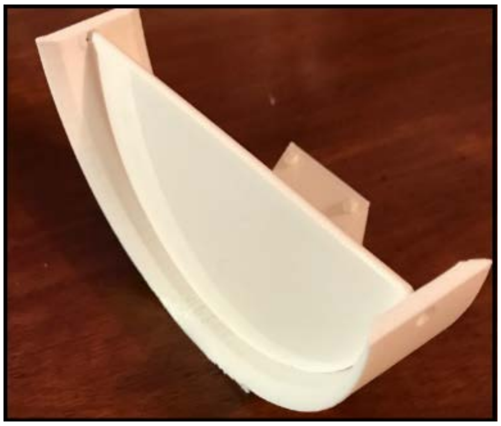


Figure 5.5.4: Closed vane flapper

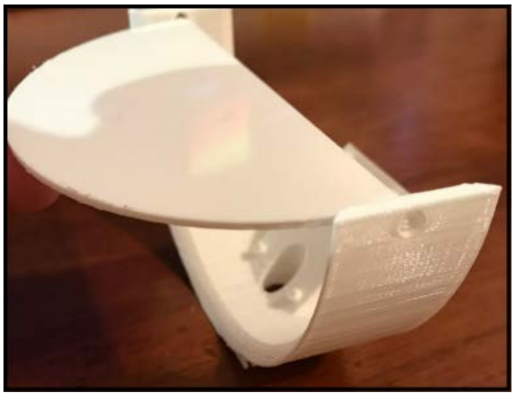


Figure 5.5.5: Open vane flapper

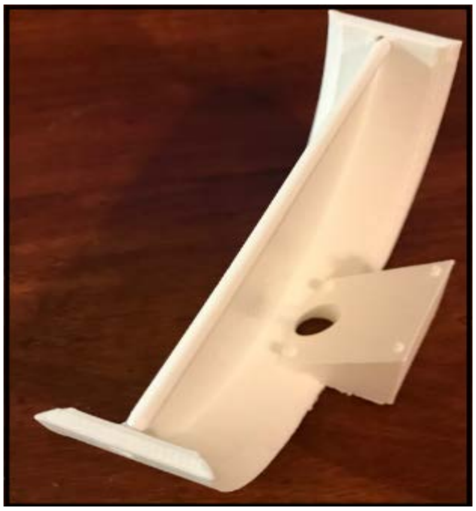


Figure 5.5.6: Back view of sensor housing

5.6 Printed Circuit Board *(Stacy Dalzell)*

The two-part design of the project required two PCBs to be designed. The first PCB would contain the processor and associated circuitry. The second would contain both the temperature sensor and the IR distance sensor for the airflow sensor.

5.6.1 Processor PCB

The processor PCB incorporates the following sub-systems and interfaces.

- Voltage Regulation.
- Microcontroller.
- SD Card.
- Voltage level shifter.
- Programming connector.
- Battery connector.
- Sensor board connector.

The complete schematics for the processor PCB are shown in Appendix B. A four-layer PCB design was selected, which enabled the 3.3V and the ground power planes to be placed on the internal layers of the PCB. This resulted in a smaller PCB and helped to meet the size constraint of the processor and battery enclosure. To further reduce the size of the PCB surface mount components were used.

To incorporate the sub-systems into a signal system, the interfaces between the systems had to be designed. The IR distance sensor operated with a 2.8V logic level, while the microcontroller was a 3.3V device. For the IR distance sensor to operate correctly, voltage level shifting was implemented. The Texas Instruments TXS0104E-Q1 4-bit bidirectional voltage level translator was incorporated. This device shifted the logic levels for the I2C and IR distance sensor enable lines.

To enable the sensors to be left in place during data recovery and battery change, a cable and connector were required between the PCBs. For cost reduction and availability, Cat 5 Ethernet cable and RJ45 connectors were selected. This selection was able to support the three power connections and the five data connections.

5.6.2 Sensor PCB

The sensor PCB was only required to incorporate the IR distance sensor and the temperature sensor. This reduced component count allowed a two-layer PCB design to be selected to reduce cost. To further reduce cost and profile of the assembled PCB, the Cat 5 cable was soldered directly to the PCB, thus eliminating the need for a second RJ45 connector jack.

The mounting pads for the temperature sensor were placed on an area isolated from most of the PCB. This is shown in Figure below. This is an attempt to reduce a heatsink effect on the temperature sensor. Optimally, this should increase accuracy and reduce the response time of the sensor.

6 Testing Methods

In this section, elaboration is made on the methodology used for testing individual subsystems. A detailed description for the construction of test fixtures, testing processes, and accuracy verification for both individual components and assembled systems is given.

6.1 Airflow Sensor *(Connor McCurley)*

The VL6180X was attached vertically to a small wooden box using double sided tape, such that it was taking measurements horizontally. This box was placed atop double-sided tape on a flat, horizontal surface. This prevented the sensor from falling out of alignment during testing. The sensor was connected to an Arduino mega, using the open-source code shown in Appendix C2 to take measurements, the readings of which were shown in real time on a serial monitor. From the base of the sensor, distances were marked on the surface at millimeter intervals. A piece of solid white printer paper was moved along these intervals and the distance measurements taken by the VL6180x were observed. The sensor maintained millimeter accuracy between 0-100 mm (3.9371 in). Since the vane flapper setup could not rotate past 3 inches, this resolution was deemed adequate for the project.

To test the operation of the airflow sensor sub-system, a test fixture consisting of 4″ ventilation duct connected to a duct fan capable of producing airflows up to 1200 FPM was created. A pitot-static tube sensor was installed in the side of the ductwork and powered by 9VDC, supplied by a lab power supply. This configuration is shown in figure 6.1.1.

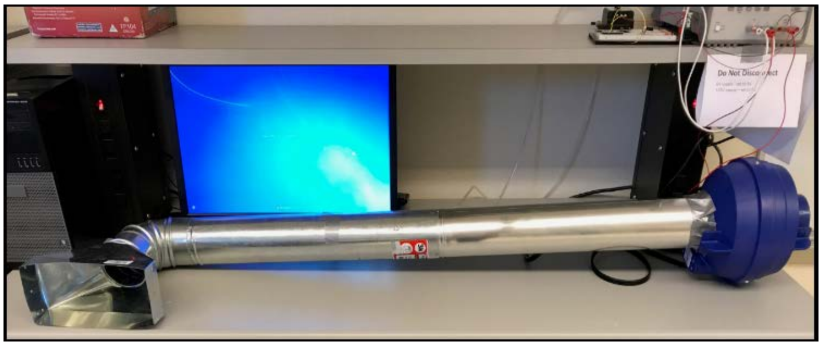


Figure 6.1.1: Airflow sensor test fixture

The vane flapper was installed with a VL6180X and the fixture was placed inside the duct outlet. The Vl6180X was connected to an Arduino Mega, similar to the distance accuracy test. The pitot-tube sensor was connected to an analog input on the Arduino. The code shown in Appendix C2 was utilized to simultaneously collect data for the position of the vane flapper, as well as the airflow data from the pitot sensor; by blocking the back of the fan and slowly allowing more air into it, it was possible to obtain data from 0 FPM to 1200 FPM. The sensor data was stored to an SD card in a .txt file. It was then possible to import the data into Microsoft Excel and obtain the best fit line for the data set. This is shown in figure 6.1.2.

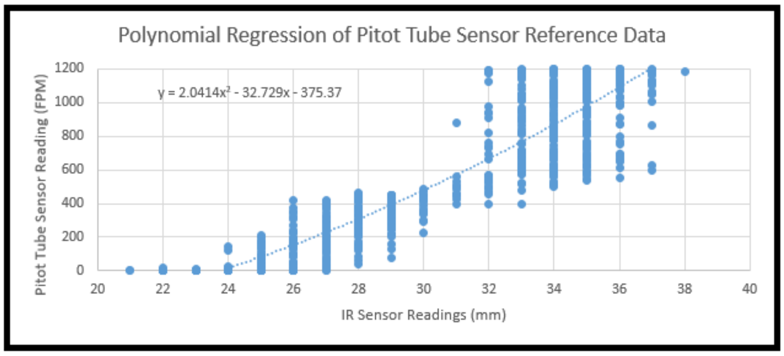


Figure 6.1.2: Best fit line for airflow sensor

This equation was implemented into the Arduino code as

$$airflow\ velocity_{FPM} = 2.0414(distance_{mm})^2 - 32.729(distance\_mm\,) - 375.37.$$

For each distance measurement taken by the VL6180X, the calculated airflow velocity could now be generated.

The test procedure was again run and the vane's position, calculated airflow, and pitot-tube airflow were recorded for 912 measurements. The results are shown in section 7.1.

6.2 Temperature Sensor *(Stacy Dalzell)*

Verification of the temperature sensor accuracy, response time, and power consumption were required. Power consumption was measured when the entire system was tested to determine total power consumption. The sensor was observed under varying temperatures to determine sensor response. The results of these tests and will be discussed in the Results section below.

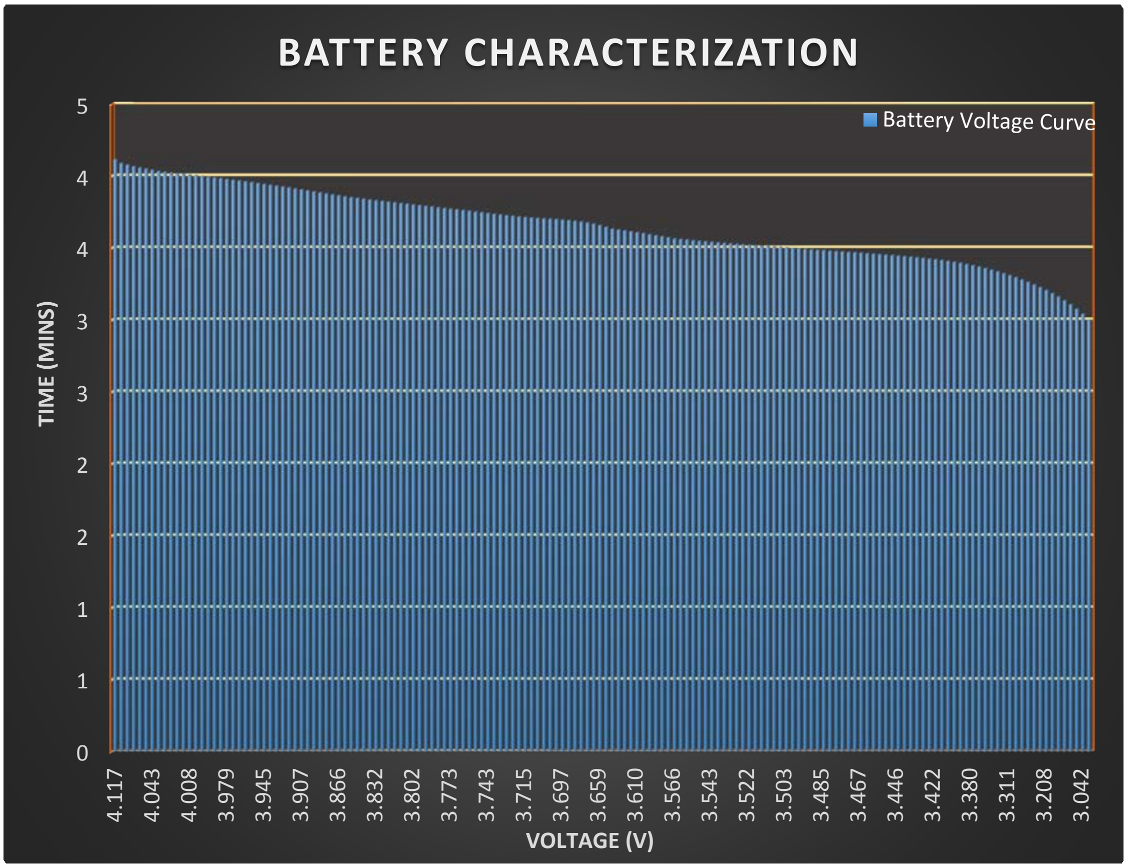
The accuracy testing required either a known temperature source or an accurate thermometer with a similar response time. An accurate thermometer was not available and the budget was insufficient to purchase one, a distilled ice water bath was created. A waterproofed sensor PCB was placed inside the ice water bath; after the temperature stabilized, temperature output from the microcontroller was read and a voltage measurement was taken at the ADC input.

6.3 Power *(Aaisha Khalid)*

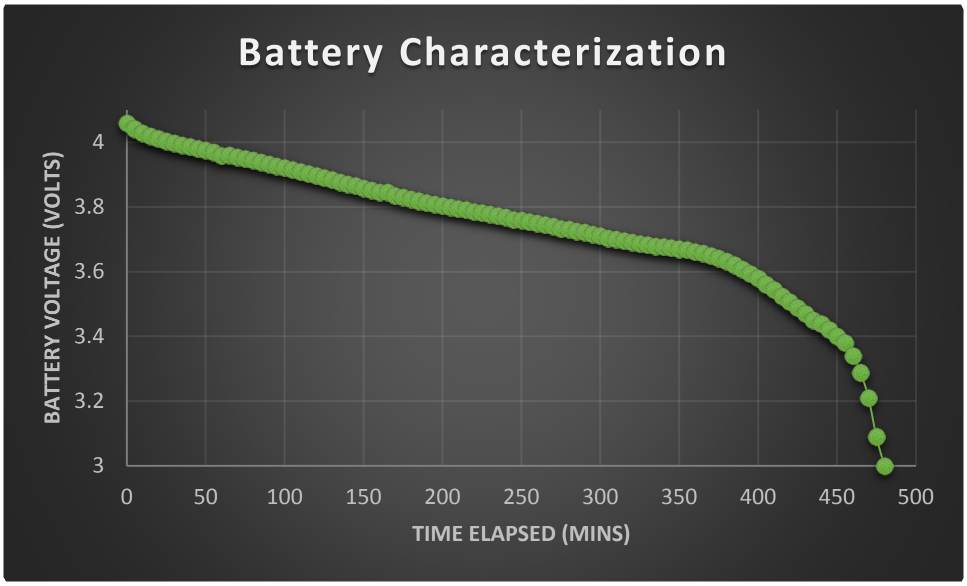
6.3.1 Battery

A battery characterization was done to calculate the drop out voltage and capacity of the battery. The characterization was done on 2 different batteries to confirm the characterization pattern.

A constant current load of 250 mA was applied to a charged battery. As time elapsed, the nominal voltage decreased. The drop out voltage was observed between the constant decrease in nominal voltage and sudden decrease in voltage. Nominal voltage of the first battery slowly decreased to 3 volts in the span of 7-8 hours; however, as time elapsed, the nominal voltage decreased more rapidly. The two graphs shown below represent two different but very similar characterization curves.



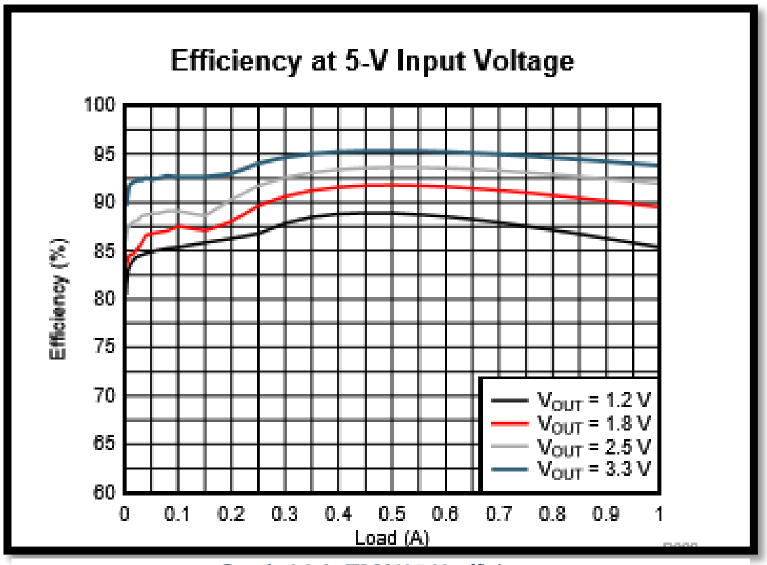
Graph 6.3.1: Battery characterization 1



Graph 6.3.2: Battery characterization 2

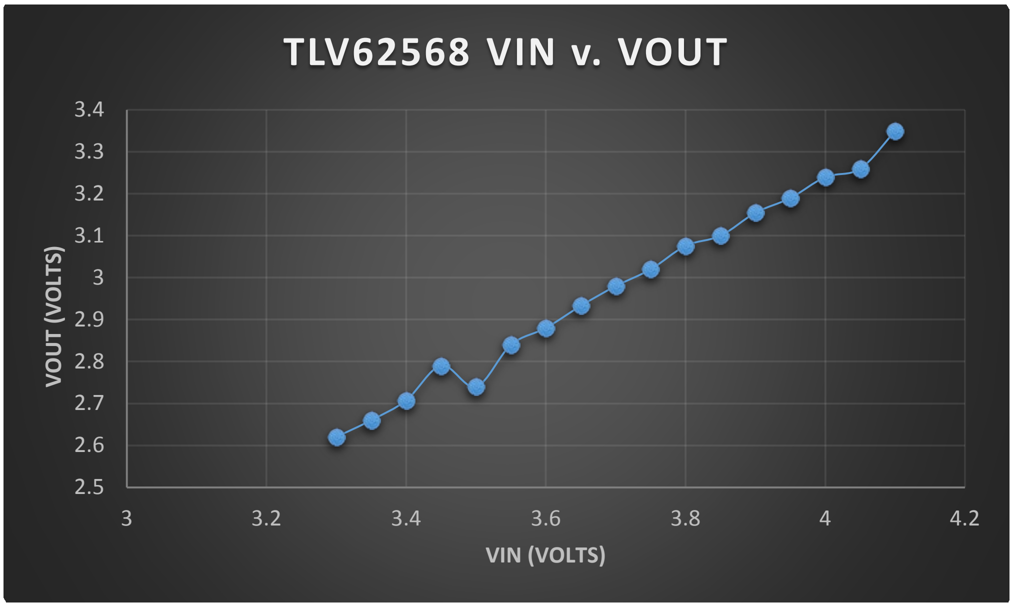
6.3.2 Voltage Regulators

Voltage regulators were tested using breadboarding and troubleshooting. First, a buck converter (TLV62568) was tested to determine the operational efficiency of the regulator. The TLV62568 datasheet contained efficiency curves that are shown in the graph below. As shown, efficiency of Vout at 3.3V is about 95%. According to the datasheet, the efficiency of Vout is between 80 and 95% if the input is within 1.8 and 5.5V [3].



Graph 6.3.3: TLV62568 efficiency

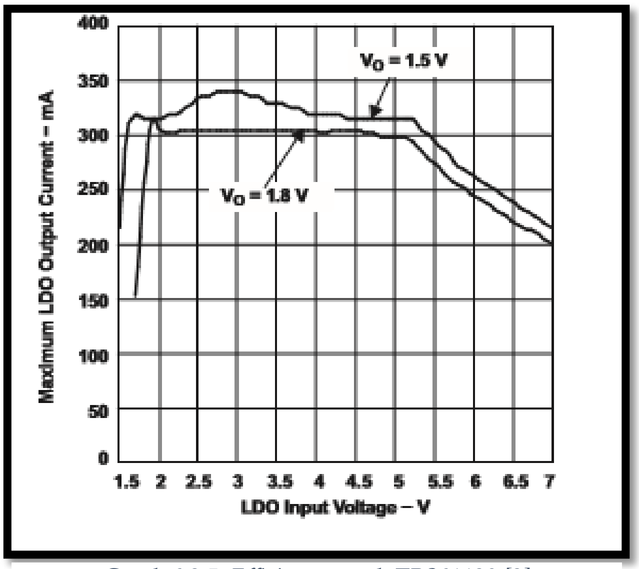
The Vin vs. Vout graph of TLV62568 below shows a sharp decrease in the efficiency of the output. The desired output of 3.3 V for the airflow system is achieved momentarily as the output voltage decreases significantly and the input voltage decreases constantly.



Graph 6.3.4: Vout graph TLV62568 regulator

The TPS61130 regulator was tested using a similar method used to test the efficiency of TLV62568. The output current and input voltage graph from the data sheet was used to verify the trend of the output current.

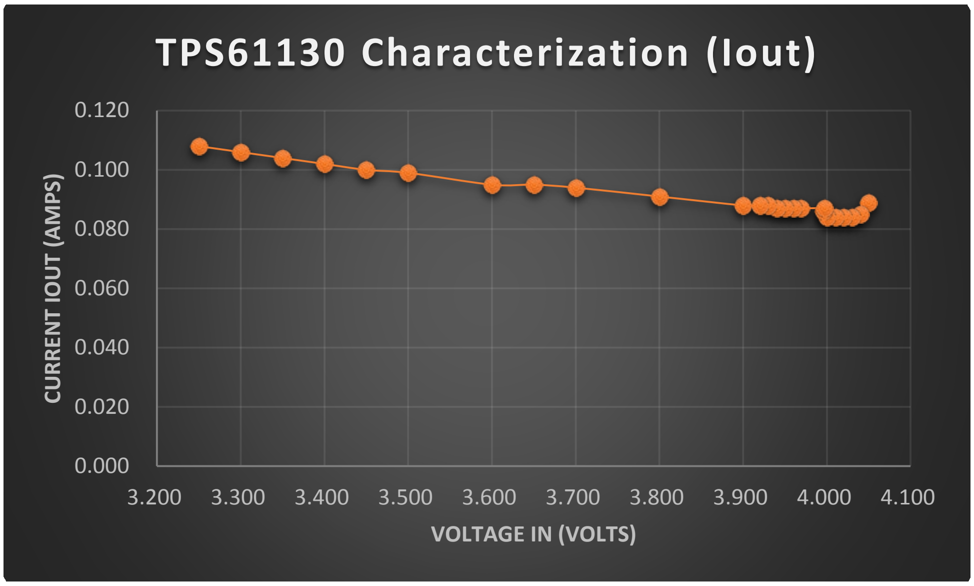
There are two outputs available in the TPS61130 regulator. One output supplies 3.3 V and the second (LDO) output supplies 2.8 V. Accuracy of the output voltage was tested using the LDO output. The graph shown below shows the output current of the LDO output of the regulator [4].



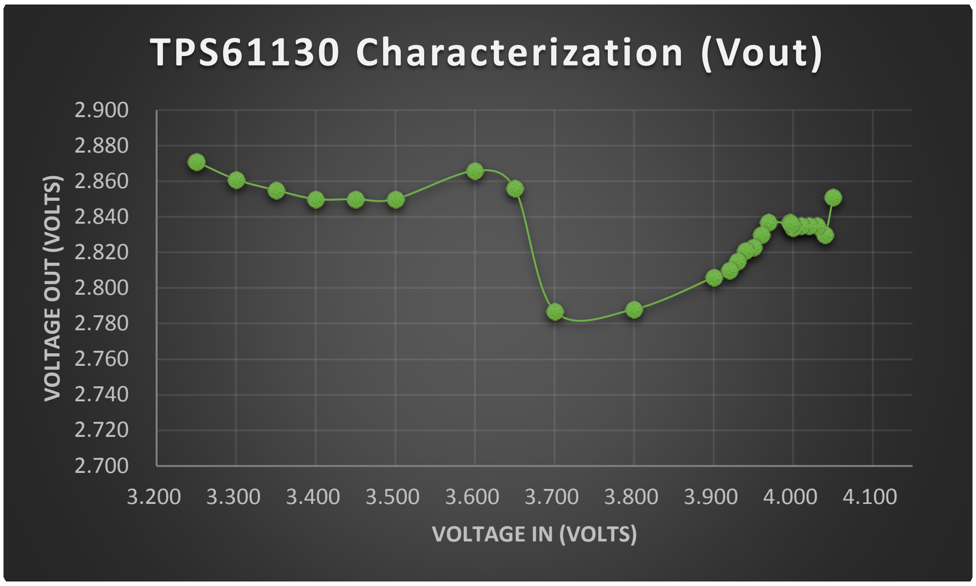
Graph 6.3.5: Efficiency graph TPS61130 [3]

As seen in the graph below, the output current is less than that of the data sheet. However, the output current and voltage graphs have a constant trend.

As shown in the schematic earlier, the LDO output supplies 2.8 V if it is provided 3.3 V from the first output. The results satisfied the power specifications. Therefore, the TPS61130 regulator was used to provide regulation for the power source.



Graph 6.3.6: Iout graph of TPS61130



Graph 6.3.7: Vout graph TPS61130

6.4 Processor *(Kevin Hill)*

Initially, a simple boot-loader program from STMicroelectronics and a USB to Serial Cable to program the processor. When no connection developed between the program and the processor, a ST-Link V2 programmer was utilized. This programming device allowed for successful communication through STM's ST-Link software tool, allowing the processor to be programmed through a hex file. In order to generate a hex file, the group used a software tool from ARM called KEIL. KEIL is an integrated development environment (IDE) made specifically to program ARM processors like the Cortex M0. KEIL was used to translate code written in C to a corresponding hex file which can then be uploaded to the Cortex M0 using the ST-Link tool. LED blinking functions were used to confirm that a successful programming protocol had been established, and development could continue.

To test the various systems once connected to the processor, the most useful tool was the universal asynchronous receiver/transmitter (UART) port. Using UART, the group could view data read from the sensors and subsequently test it for accuracy. UART was also used to test data conversion algorithms and to confirm code was successfully passing through created functions and not getting stuck in infinite while loops or other traps.

6.5 Printed Circuit Board *(Stacy Dalzell)*

Testing of the PCBs was primarily accomplished by the individual sub-system testing procedures. The overall operation of the complete system was monitored for correct operation as a second verification that the system was operational.

7 Results

This section provides tangible evidence to the adherence of individual subsystems to their respective design specifications. Elaboration of results is given to explain the significance of all findings.

7.1 Airflow Sensor *(Connor McCurley)*

IR Sensor Accuracy:
It was found that the VL6180X distance ranging sensor was accurate to the millimeter. This implied that we could read variations in the vane flapper's position with resolution to the millimeter.

7.1.1 Timing

To meet the specifications of this project The airflow sensor was required to wake up, take a single reading, and go back to sleep in a time period of under 30 seconds. This requirement was narrowed to take a single reading per second. The VL6180X, however, could take approximately 800 readings per second; therefore the project requirements were fulfilled.

7.1.2 Airflow Restriction

The maximum airflow velocity through the test fixture before the installment of the vane flapper was 1210 FPM. After installment, the maximum velocity in the test fixture duct reached 1208 FPM. A difference of 2 FPM can be considered negligible for our purposes. This implied that there was almost no air restriction with the implementation of our airflow sensor sub-system. This met the restriction requirements for the project.

7.1.3 Duct Alteration/Dismantlement:

It was not necessary to dismantle or alter the air duct in any way to install or operate our airflow sensor sub-system. This met the specifications for the project.

7.1.4 Sub-system Accuracy

Due to the design of the vane flapper, we could only utilize a range of 20mm from the VL6180X distance sensor. An airflow of zero FPM was detected between 20-23mm, and the maximum airflow of 1200 FPM was detected at a vane flapper position of 40mm.
The figure below shows the correlation between our reference (pitot-static) airflow readings and the position of the vane flapper for a set of 912 measurements.

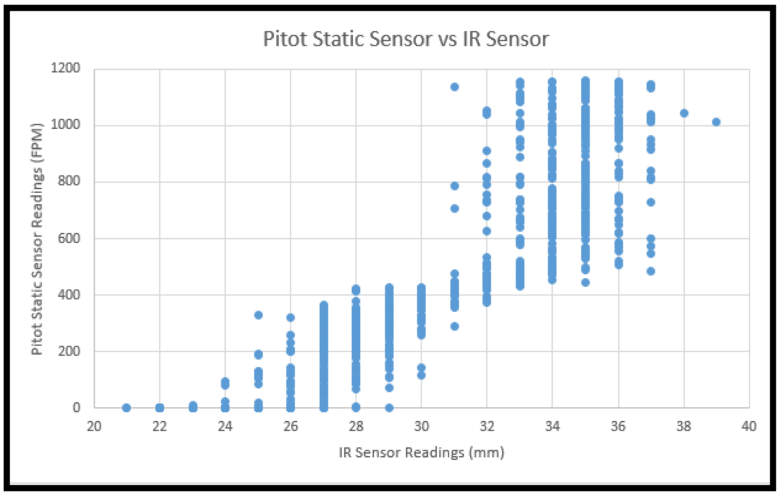


Figure 7.1.1: Reference airflow velocity against vane flapper position

It can be seen that for a multitude of distances read up by the VL6180X distance sensor, the pitot-static sensor outputs a range of values with variances anywhere from 400 FPM between 25 and 29 mm, to 700 FPM above 31 mm.

The representation for the percent error between the pitot-static reference data and user-defined airflow sensor characterization is shown below. This dataset acknowledges known irregularities caused by non-laminar flow within the duct.

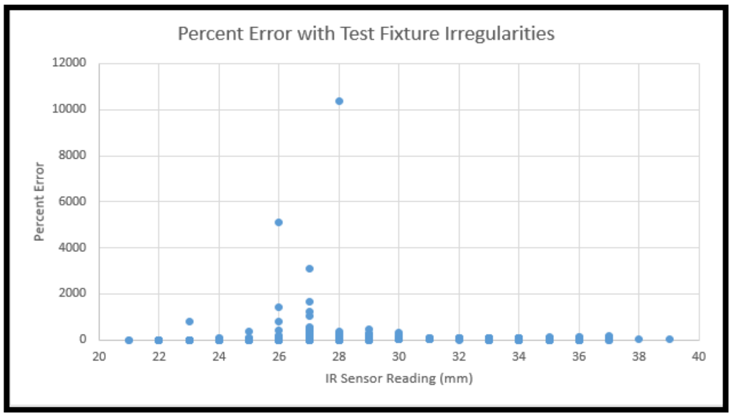


Figure 7.1.2: Percent error with test fixture disturbances

Disregarding these known irregularities, the data set for the percent error becomes the following.

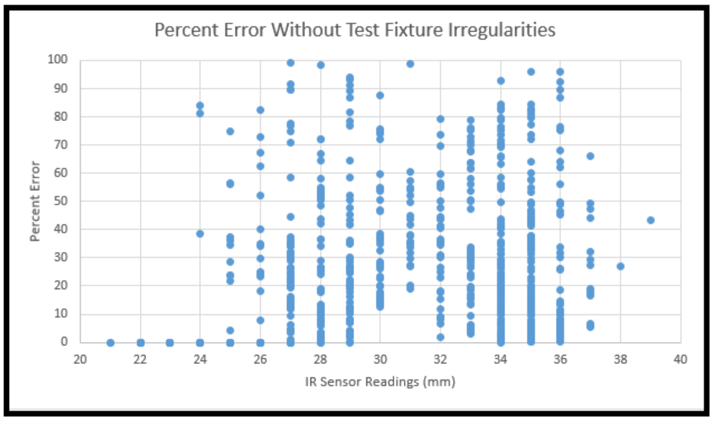


Figure 7.1.3: Percent error without test fixture disturbances

By disregarding the known irregularities of the test fixture, it can be seen that for most positions of the vane flapper, the difference between the calculated airflow and the exact airflow ranges between near zero error to 100% error.

To better understand the symbolism of this set, the percent error distribution is shown below.

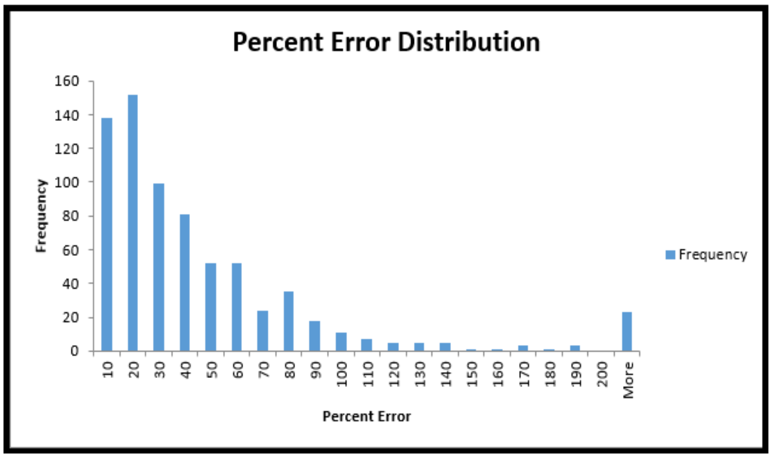


Figure 7.1.4: Percent error distribution

This implies that, while the percent error does fluctuate drastically for most positions of the vane flapper, the actual percent error for any singular reading will likely fall between 0 and 40%.

With this model, statistically, we will obtain an airflow reading accuracy error of 10% or less for 20% of our readings.

This implies that, while we are obtaining readings within our required accuracy 20% of the time, we cannot explicitly state that we meet our airflow sensor accuracy requirements for the project.

7.2 Temperature Sensor *(Stacy Dalzell)*

After final testing, it was determined that the temperature sensor met all design goals. Since the total power consumption was approximately 25% of the maximum design allowance, the temperature sensor goal was satisfied. The temperature output from the microcontroller was 0° C, using the voltage to temperature equation from the datasheet. This value was verified by taking a voltage measurement at the ADC input. This voltage was 500 mV and it matched the datasheet output voltage at 0° C. Temperature response observations noted the response time was much less than 30 seconds. During operational temperature testing, the response time of the sensor was fast enough to detect the cycling of the heat source element.

7.3 Power *(Aaisha Khalid)*

7.3.1 Battery

Calculations were needed to determine the battery capacity for the power source. The equations and calculations below prove that the battery selected to supply power to the airflow system is in fact an appropriate selection.

The average current calculations were completed once the processor achieved wake and sleep states with both sensor systems [7]

$$Capacity\ (Ah) = \ Average\ Current\ (\ in\ Amps)\ x\ Time\ (in\ hours)$$

$$0.01204\ (Ah) = \ 0.000501(\ in\ Amps)\ x\ 24(in\ hours)$$

$$1.98396(Ah) = \ 0.000501(\ in\ Amps)\ x\ 3960(in\ hours)$$

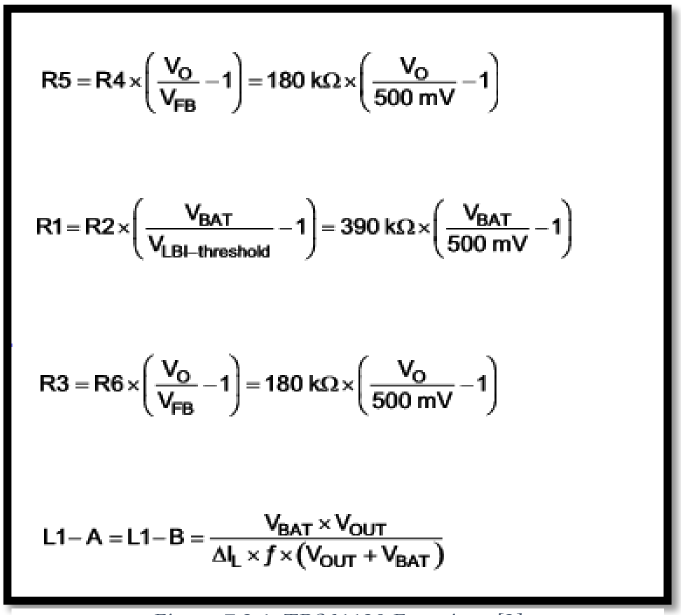
$$Avg\ Current = (Current(wake)xtime(wake)\ )/(3600\ seconds\ per\ hour \\ + Current(sleep)xtime(sleep)/(3600\ seconds\ per\ hour))$$

$$0.000501\ amps = (0.00096\ Amps\ (wake)x\ 1\ sec)/(3600\ seconds\ per\ hour) \\ + (0.0005\ A(sleep)x\ 29sec\ )/(3600\ seconds\ per\ hour)$$

The average current calculations shown above were used to check the capacity of the power source with respect to the specifications [7].

7.3.2 Voltage Regulator

The circuit component values were calculated using the equations in the data sheet. As mentioned before, the desired outputs for the regulator outputs were 3.3V and 2.8V.



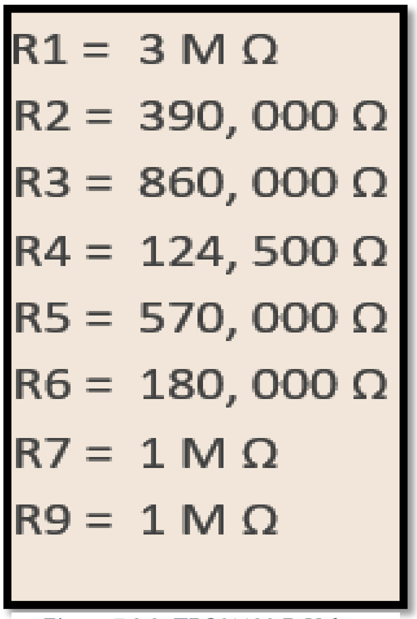
$$R5 = R4 \times \left(\frac{V_O}{V_{FB}} - 1\right) = 180\ k\Omega \times \left(\frac{V_O}{500\ mV} - 1\right)$$

$$R1 = R2 \times \left(\frac{V_{BAT}}{V_{LBI-threshold}} - 1\right) = 390\ k\Omega \times \left(\frac{V_{BAT}}{500\ mV} - 1\right)$$

$$R3 = R6 \times \left(\frac{V_O}{V_{FB}} - 1\right) = 180\ k\Omega \times \left(\frac{V_O}{500\ mV} - 1\right)$$

$$L1-A = L1-B = \frac{V_{BAT} \times V_{OUT}}{\Delta I_L \times f \times (V_{OUT} + V_{BAT})}$$

Figure 7.3.1: TPS61130 Equations [3]



R1 =	3 M Ω
R2 =	390, 000 Ω
R3 =	860, 000 Ω
R4 =	124, 500 Ω
R5 =	570, 000 Ω
R6 =	180, 000 Ω
R7 =	1 M Ω
R9 =	1 M Ω

Figure 7.3.2: TPS61130 R Values

$$VFB\ =\ 500\ mV\ (feedback\ frequency)$$

R1, R2, R3, and R6 were resistors used to produce an output of 3.3 V for output one.
R4 and R5 were resistors used to produce an output of 2.8 V for LDO.

The regulator data sheets recommended capacitor and inductor values for a range of outputs. The power engineer used the recommended values during testing and observed promising results. Since testing results were satisfactory, the values remained for the capacitors in the TPS61130 schematic unchanged throughout the project.

7.4 Processor *(Kevin Hill)*

Upon completion of the project, the processor and subsequent program received raw data from both the temperature and airflow sensors. This was confirmed by outputting results from these sensors over UART. Beyond outputting raw data, equations were developed to take the raw sensor data and translate it to usable data. The data sheets for the temperature sensor provided the following equation to translate to degrees Celsius. [1]

$$ºC = (V\_count * V\_ref - 0.5\ \ )/0.01$$

The equation to translate a distance in millimeters to the speed of the airflow is the second order polynomial equation seen below, where x is the distance in millimeters, and y is the airflow in feet per minute (FPM) [1].

$$y = 2.0414x\text{^}2 - 32.729x - 375.37$$

Once these equations have translated the raw data, the data is then translated to independent ASCII characters. These characters would then be stored in an array of characters, waiting to be sent to the SD card over SPI for permanent storage.

The group was not able to implement the functions to write this data to the SD card. This will be further addressed in the discussion section (8.4) of this report.

Sleep functions were implemented to conserve power. The Cortex M0 has several sleep modes, including low-power run, low-power sleep, stop, sleep, and standby. The STOP mode was selected to preserve register contents to prevent having to reinitialize all peripherals on wakeup. A 30 second timer is set, during which the external oscillator runs the

RTC. Once the timer is up, an interrupt handler wakes the processor and begins the measurement process again.

7.5 Printed Circuit Board *(Stacy Dalzell)*

Three component footprint errors were noted in the assembly and testing of the PCBs. Both the micro SD card holder and the battery connector had solder mask covering some mounting pads. This was easily corrected by scraping the solder mask off the pads. The third error was found during the power supply verification testing. During this testing, it was noted the 3.3V regulator output was low. Troubleshooting found the silkscreen polarity marking for capacitor C11 was incorrect. This resulted in the capacitor being installed backward during assembly. The capacitor was reinstalled correctly, and the power supply functioned accordingly.

8 Discussion

In this section, a summary of the results obtained through the testing of each individual sub-system will be provided. A thorough analysis will be completed to express the implications of the findings to the project as a whole. Clarification is offered to the shortcomings of any system component. Setbacks are acknowledged and methods for perseverance are observed.

8.1 Airflow Sensor *(Connor McCurley)*

As noted in the results section, the airflow sensor sub-system met all project requirements, including air restriction, measurement time, and duct alterations; however, the airflow sensor failed to fully meet its accuracy specifications. This can be attributed to three main reasons, 1) irregularities in the test fixture/ reference data, 2) resolution of the distance sensor, and 3) the sub-system designer lacking a sufficient background in fluid dynamics.

8.1.1 Test fixture turbulence

As seen in the results section, we obtained a percent error characterization between our calculated airflow and our reference airflow as the following:

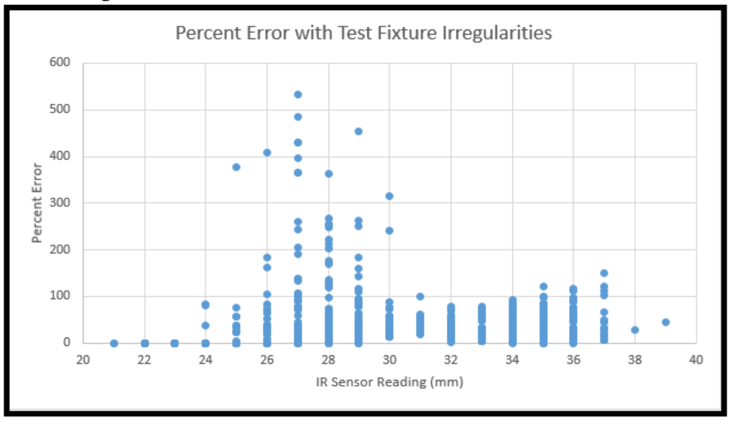


Figure 8.1.1: Percent error with test fixture disturbances (zoomed in)

From evaluating the measurements, it was observed that there were "divots" in the data. In other words, the airflow would drop to 0 FPM for one measurement then spike to a reading well outside the range of its neighbors. This in turn, would cause the percent error calculation to soar well outside the expected range. On the graph above, these points are represented as anything above 100 FPM. These dips and jumps can be attributed to turbulence in the test fixture duct, caused by the breakup of laminar flow. To ease the effects of this turbulence, a makeshift flow straightener was made out of a bundle of drinking straws. The flow straightener, as the name suggests, attempted to take turbulence out of the flow stream. While this did aid in reducing air turbulence in the test fixture, it was not significant enough to eradicate the irregularities in the test data. Knowing that these disruptions would not exist in a better constructed test fixture, it was assumed these nuances were negligible for the airflow sensors accuracy analysis. Future development necessitates more care be taken in the construction of the test fixture to eliminate discrepancies caused by turbulent flow.

8.1.2 Distance Sensor Resolution

The main contributing factor to the inaccuracy of the airflow sensor comes from the resolution of the VL6180X distance sensor. In testing, it was discovered that the VL6180X is capable of accurate ranging to the millimeter. During design, the extent of displacement the flapper would undergo with variations in airflow was greatly overestimated. It was found in testing that for some airflow velocities, there could be a change of up to 400 FPM before the flapper was displaced a full millimeter. To reiterate, the calculated airflow velocity comes from the equation,

$$airflow\ velocity_{FPM} = 2.0414(distance_{mm})^2 - 32.729(distance_{mm}) - 375.37$$

This implies that there is only one expected airflow velocity for each integer distance between 20 and 40mm, while in fact, the distance is contained in an epsilon neighborhood around the calculated airflow value. This introduces a large margin of error. It is imperative for future development that a distance sensor be selected with greater resolution, as to improve the calculated approximation and reduce error between the calculated and exact airflow value.

8.1.3 Background

The third attribute which contributed to airflow sensor inaccuracy was the airflow sub-system designer's lack of theoretical background and working knowledge of fluid dynamics and related mechanical systems. A limited knowledge of fluid dynamics caused Connor to overlook various aspects, such as the actual displacement the flapper would undergo with minuscule changes in flow velocity. This in turn affected the choice of distance sensor resolution. Application of Computational Fluid Dynamics would be efficacious to future development in determining flapper dynamics, modeling and eradicating test fixture turbulence, and finding the appropriate method of damping to smooth measurements taken by the airflow sensor sub-system.

Due to difficulty in programming the microprocessor, the timeline for testing the airflow sensor was significantly reduced, which resulted in a short implementation period to debug the sensor and test fixture.

These factors contributed to the inability to meet the airflow sensor accuracy requirements outlined by the project specifications. While the prototype system did not meet all the regulations this semester, it has been shown that with a more accurate distance sensor and a theoretical development of the system's parameters, the vane-flapper method of airflow sensing is capable of fulfilling all specifications as described for the project.

8.2 Temperature Sensor *(Stacy Dalzell)*

The main limitation to the temperature sensor characterization was the lack of an accurate thermometer and a controllable heat source. This limited the accuracy of the sensor and necessitated the use of a known physical constant as a temperature source. An ice water bath was created using frozen distilled water. Using the freezing point of distilled water of 0° C, the accuracy of the temperature sensor was verified. The observed output of 0° - 0.25° C from the microcontroller was within the specification for temperature sensor accuracy.

Further characterization of the temperature for a production device is needed. Characterization will need to identify the measurement errors over the entire expected operating range. This data would be used to improve the accuracy of the manufacturer's typical output voltage characterization.

8.3 Power *(Aaisha Khalid)*

Power specifications for the project were surpassed. Battery for the project proved to be an excellent decision. The additional power and the regulator's adaptability proved instrumental while the team was performing troubleshooting. Optimization of battery performance may be achieved by obtaining a battery with higher battery capacity. Overall, the performance of the regulator exceeded team's expectations and minimum troubleshooting was required.

8.4 Processor *(Kevin Hill)*

The STM32CubeMX software tool by STMicroelectronics was selected to handle most of the low level initialization functions and pin mapping. This tool allowed the group to map each pin as desired and to control the core and real time clock sources. It also initialized the necessary peripherals, such as SPI, I2C and the ADC, and provided functions to transmit and receive data from them [3].

Once low level functions were created, Attention was paid to a method to determine the peripherals were working which is where UART comes in. UART, when combined with a simple serial monitor, such as RealTerm, allowed for easy data reading and debugging. The UART functions provided by STM32CubeMX themselves proved quite simple to read and understand; however it presented an unknown baud rate and an unknown number of data bits. RealTerm defaults to a baud rate of 57600 and 8 data bits, leading to unreadable data being displayed. This problem was corrected by simple trial and error in the settings of RealTerm, leading to the required baud rate of 115200 and seven data bits.

Values being received from the sensors were tested once the UART was functional. The ADC was tested for accuracy by comparing the voltage value being displayed over UART to the true value being displayed by a multimeter. The error margin for the ADC had a typical value of 0.05mV which is explained by the specified voltage resolution of

$$\frac{V_{ref}}{2^{12}} = \frac{3.3}{4096} = 0.0806mV.$$

Data was read over I2C from the VL6180X distance sensor; however, the sensor had multiple modes of operation to return different values based on a desired purpose for the sensor. This entails initializing the sensor with 39 instructions provided by STMicroelectronics, the chips manufacturer. The second complication was that the instructions provided by the manufacturer were in a library built for an Arduino. The group had to translate this library's functions over to properly initialize the sensor. One last problem came from a communication failure flag on the Cortex M0. If the Cortex M0 failed to communicate with the VL6180X, a flag would be set causing the Cortex M0 to stop attempting to communicate over the I2C bus. The only solution the group discovered to solve this issue was to restart the VL6180X, and reinitialize the I2C bus [3].

Researching and testing the sleep modes already defined by the Cortex M0, the group discovered there are five of these modes. These modes are low-power run, low-power sleep, stop, sleep, and standby. The lowest power consumption of these modes is the sleep mode. While it is significantly lower power consumption, it was declared unviable because it loses power to Cortex M0's SRAM, which clears all registers. Clearing all registers means that the M0 cannot be woken up by interrupt, and can only be woken up by activating a pin externally, which means increased external circuitry based on a timing system. With all registers cleared, all peripherals including the core clock would have to be reinitialized as well. The next lowest power mode is the stop mode, which is implemented in the final design. This mode is chosen because it does not clear register contents, enabling it to be woken by interrupt. While stop mode does shut down the M0's internal oscillator, a simple external oscillator circuit keeps the real time clock running and allows for a wake up timer.

The group was not able to implement SD card writing functionality as hoped for in this design. The Cortex M0 offers support for the FATFS library which is required to format and store data on SD cards and be readable on many devices including Windows PCs. However, this library still requires low level peripheral controls to communicate successfully over SPI. These controls include timing to set SD cards into SPI mode and paging systems to transfer data in a way that can be stored by the SD card. Future development would have to develop a way to point the FATFS library to the SPI and send data correctly. As proof of concept, there is an Arduino library already made for sending information to an SD card over SPI and the group used this library when gathering data to formulate an equation for airflow. A research on the complexity of all systems to ensure that all base functionality can be implemented in the given amount of time for a project. The Arduino could have been implemented to complete this aspect of the project. However, the group decided early in the design process to not use the Atmega chip due to the Cortex M0 is a lower power device which was priority for this project.

8.5 Enclosure

Restrictions for the enclosure space were not as constraining as other specifications of the project, and it was for practical purposes, that the battery and processor board were packaged together.

8.5.1 Processor Enclosure (Aaisha Khalid)

The enclosures for the airflow system were printed using 3D PLA material with a printing temperature of 190-220° C, which provided temperature tolerances within project specifications. Better tolerance could be achieved with the use of ABS material, which has a printing temperature of 230-250° C. Improvement in 3D printing material would subsequently improve overall temperature specifications for large scale system production [9].

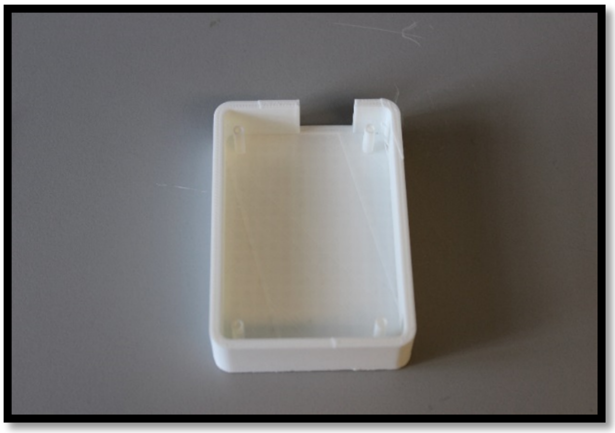


Figure 8.5.1: Top piece: Enclosure

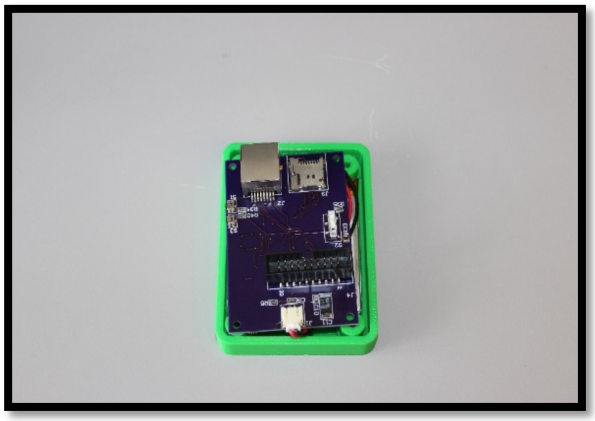


Figure 8.5.2: Bottom piece w battery & PCB

8.5.2 Sensor Enclosure *(Connor McCurley)*

The designed vane flapper structure proved to be simple and effective for gaining airflow readings. One of the concerns stated for this design was the existence of friction between the vane and sensor housing Both the vane and sensor housing were constructed from PLA plastic for the extend of this project and, as plastic tends to be abrasive on plastic, friction would be a concern if this was a less dynamic system; however, low level duct turbulence kept the flapper in a constant state of motion. This motion, while minuscule, eliminated the effects of static friction between the two components. It would be beneficial to employ metal on metal rotation or use a type of bearing to minimize the long-term degradation at the point of contact between the vane and sensor housing.

8.6 Printed Circuit Board *(Stacy Dalzell)*

The limited amount of time available for development of this design allowed for only one PCB design to be fabricated. The success of the design was due to several factors during the design process which are detailed below.

The voltage regulator, voltage level shifter, temperature sensor, and the microcontroller were assembled on breakout boards and the circuits were constructed on breadboards. These circuits were tested for function and suitability to meet project specifications. These test circuits were assembled with as much of the related circuitry as possible, such as the external oscillator circuit for the microcontroller.

Each circuit included in the PCB was reviewed to verify compliance with recommendations from manufacturer datasheets and application notes. This research allowed the circuits to be improved for reliability and to ensure proper function.

The updated individual circuit schematics for each subsystem were compiled into two final schematics. The first schematic included the processor board systems, while the second included the sensor board systems. Individual connections between components were verified to be correct.

The PCB footprints for the components were created in Altium designer once the schematics passed design review. These footprints were verified with the datasheets for the device and by printing the footprints at actual size and physically comparing with the components.

The PCBs were designed to minimize the size of each board. This was accomplished by using surface mount components and careful placement of components. The positioning of components was optimized to minimize trace length and number of vias. This allowed the components to be placed closer together resulting in a more compact PCB design.

The design rule check feature of Altium Designer was then used to verify that the PCBs met the requirements of the PCB fabricator. This was an important step as Osh Park, the fabricator chosen for this project, does not perform design rule checks on designs submitted for fabrication. This and the factors detailed above ensured the PCBs would perform as designed.

The cost to fabricate three copies of each board design was $65.85. This cost can be reduced by using a normal PCB fabricator instead of a prototyping service. This will allow the boards to be produced in larger quantities at a more effective price.

9 FUTURE WORK *(STACY DALZELL)*

This designed can be improved in future design efforts. Some of potential improvements, such as sensor characterization and SD card functions, where discussed in the sections above. Improvements can also be made in the cost of the device as well as by improving the battery life.

9.1 Cost Reduction

The Bill of Material (BOM) cost for producing one prototype unit is detailed in Table 9.1.1 below. These were the costs incurred in the production of the electronics for three prototype devices. The low volume of components resulted in a unit cost of $60.36.

Description	Quantity	Cost
Processor PCB Fabrication	3	$58.10
Sensor PCB Fabrication	3	$7.75
Components	3 units	$76.37
Battery	1	$12.95

Table 9.1.1 Low Volume Production Costs

The unit cost of the design can be reduced by larger production runs and eliminating unnecessary components. Several components were added to the design of the prototype to facilitate programming changes, design changes, and trouble-shooting. These components can be deleted in a production device to reduce cost. Increasing the number of units in each production run will reduce the unit cost by taking advantage of volume pricing of components. A production run of 100 units was modeled to illustrate the cost savings of a larger production run. The costs are shown in Table 9.1.2 below. The cost reductions resulted in a unit cost of $34.25.

Description	Quantity	Cost
Processor PCB Fabrication	100	$88.00
Sensor PCB Fabrication	100	$18.08
Components	100 units	$2022.75
Battery	1	$12.95

Table 9.1.2: Production costs 100 units

9.2 Battery Life Improvements

The battery life was an important factor in the design and any improvements would be a priority in any future work. The battery life can be increased by optimizing the software algorithm and sleep cycle.

The software can be optimized to reduce the number of processor cycles required to complete the measurement cycle will reduce the time spent at high power states. Optimizing the order high power systems wake from sleep states may result in further power savings. Quantitative analysis can be performed of each available sleep mode to explore other power reduction options.

10 CONCLUSION *(AAISHA KHALID)*

This project sought to improve testing and measuring techniques of HVAC duct operations. Team members tackled the complex air flow sensor system by dividing the project into major subsystems. The team faced many challenges while integrating each subsystem. Software compatibility, hardware troubleshooting and inconsistent testing accuracy were some of the problems addressed by the team. Prototyping, early troubleshooting and repetitive testing solved many hardware problems within the project.

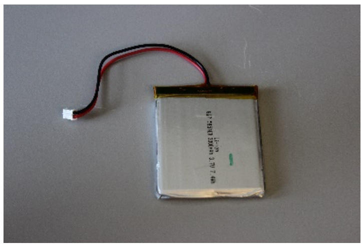
APPENDIX A *(STACY DALZELL & AAISHA KHALID)*

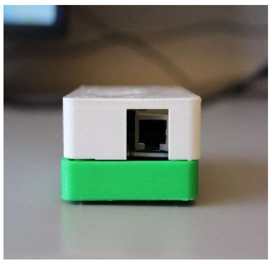
Instructions: HVAC Airflow Sensor

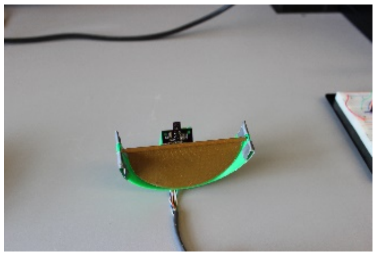
INTRODUCTION

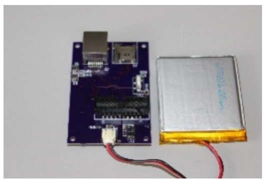
This project seeks to design an Airflow sensor system to be placed in a HVAC duct. This document provides instructions for maintenance of the Airflow sensor system.

PART IDENTIFICATION


Battery & Connector


Processor Enclosure


Airflow Sensor


Processor Board & Battery

BATTERY, SD CARD, AND PROCESSOR BOARD ASSEMBLY

1. Insert formatted micro SD card.
2. Connect charged battery to processor board.
3. Place battery in bottom case.
4. Place processor board into bottom case while aligning mounting holes with bottom case extrusions.
5. Place top case over bottom case assembly while aligning notch in top case with sensor connector.
6. Install case screws (qty. 4).

SENSOR DEPLOYMENT

1. Remove air grill.
2. Connect sensor cable to processor assembly.
3. Place sensor assembly into duct. Make sure the sensor is as close to vertical as possible.
4. Mount the processor assembly to the side of the register box using double sided tape.
5. Push the recessed reset button with a small hex key or similar tool.
6. Reinstall the air grill removed in Step 1.

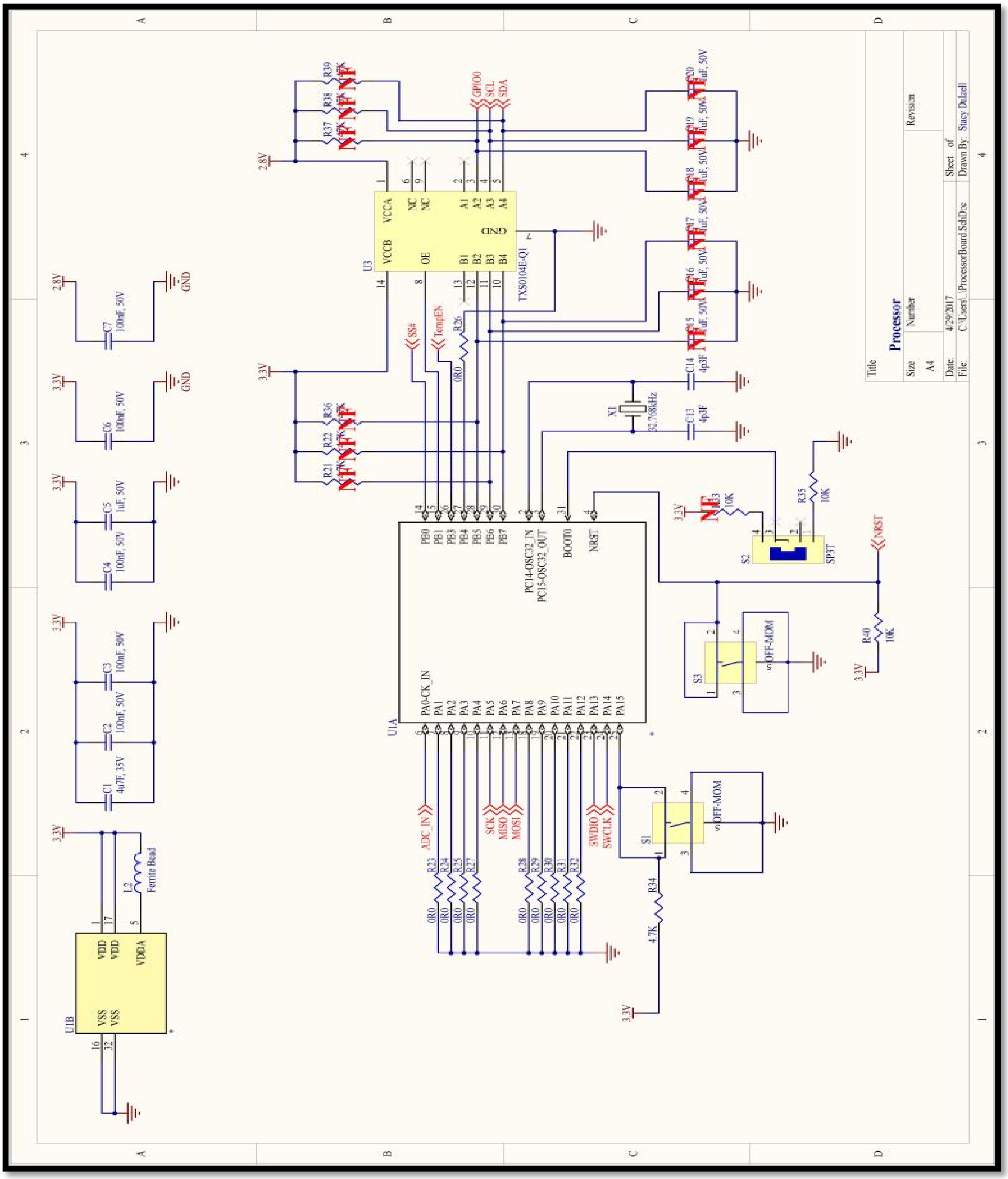
SD CARD RECOVERY & BATTERY REPLACEMENT

1. Remove air grill.
2. Press the recessed data write button with a small hex key or similar tool.
3. Remove the processor assembly from the register box.
4. Disconnect the sensor cable.
   NOTE: The sensor assembly can be removed if no further measurements are required.

5. Remove case screws (qty. 4).
6. Remove top case.
7. Lift out processor board.
8. Remove the battery from the case.
9. Remove the micro SD card.
10. Disconnect the battery from the processor board.
11. Reassemble using the assembly instructions.

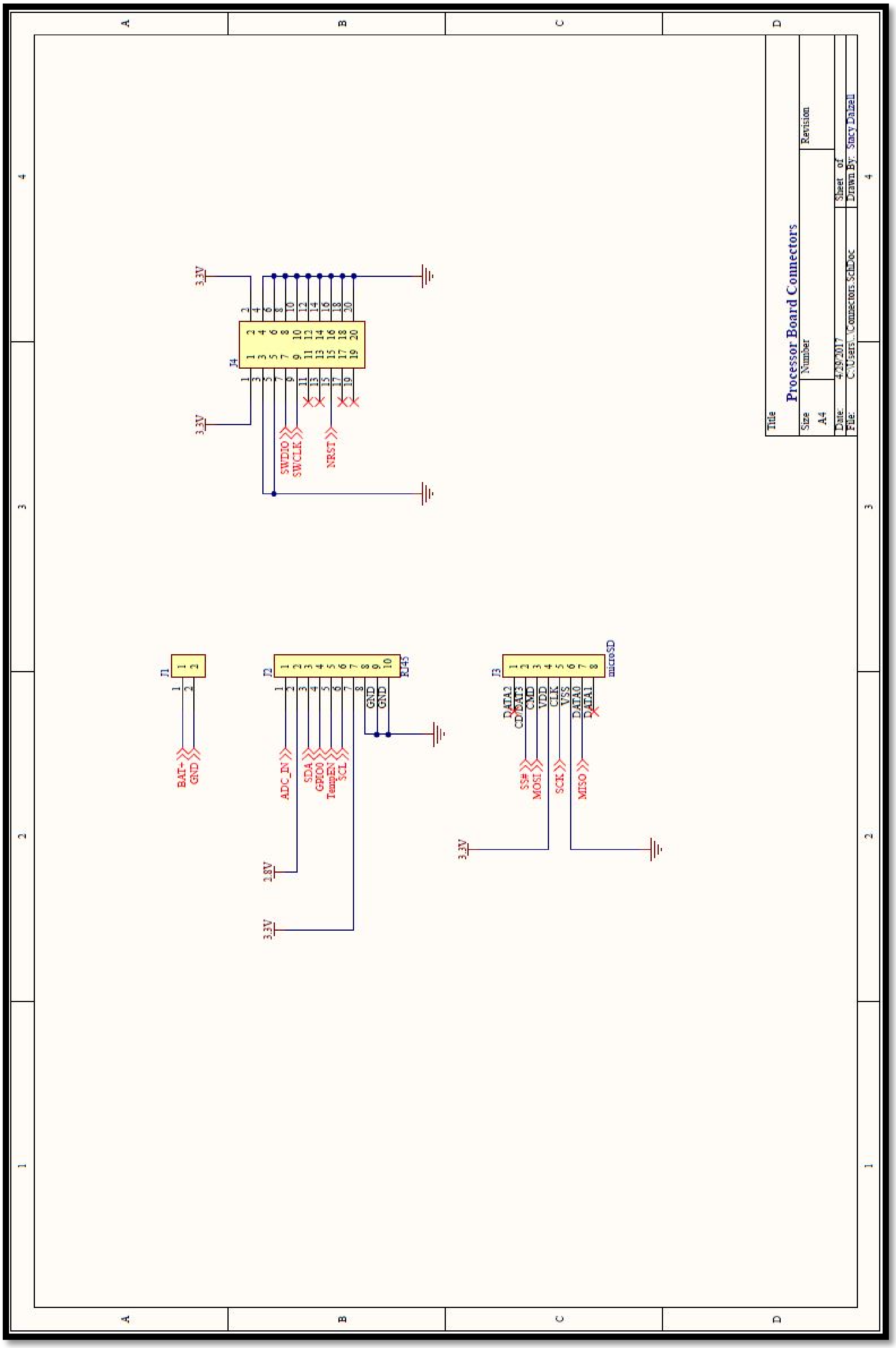
APPENDIX B1 *(STACY DALZELL)*

Processor Board Schematic



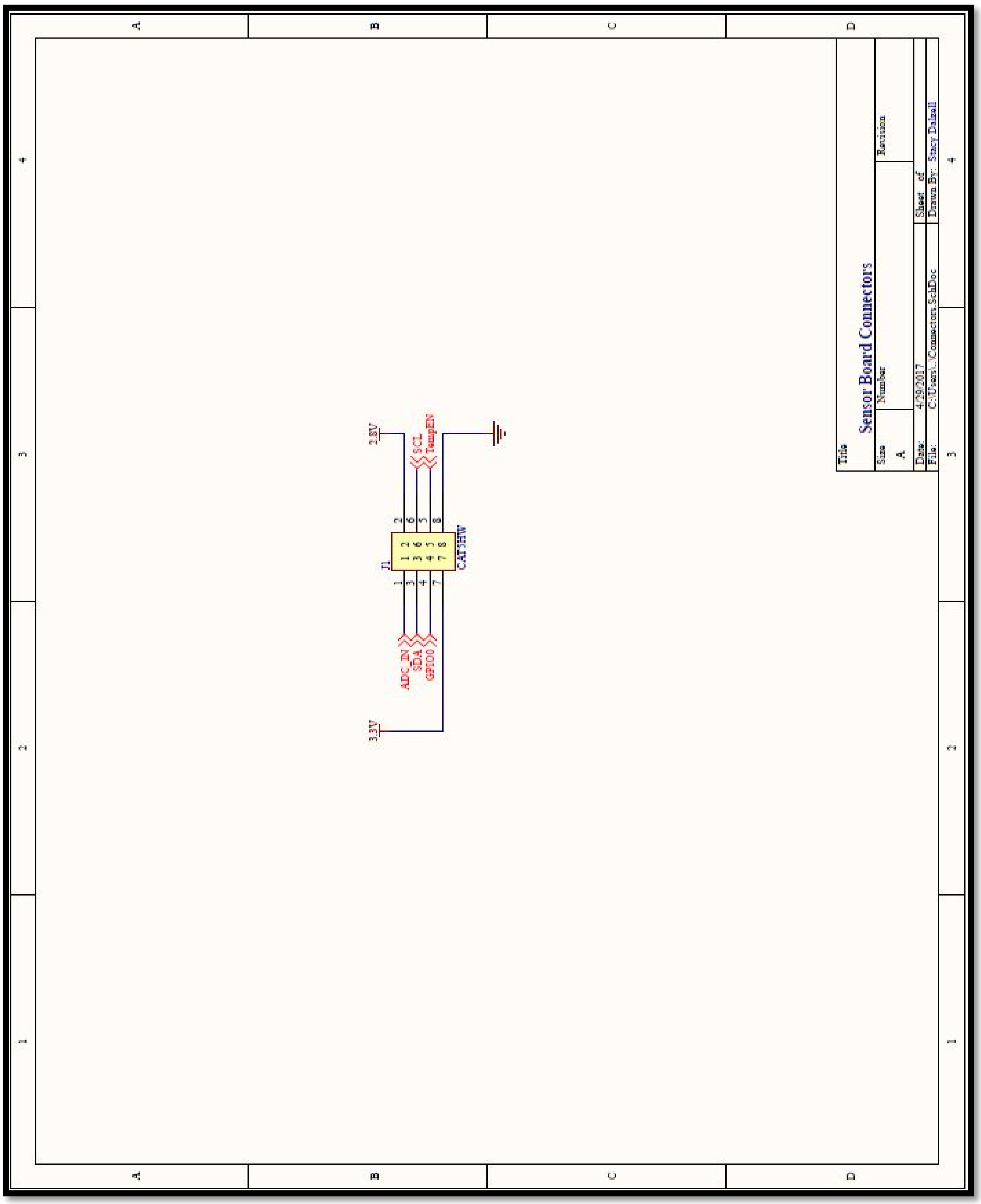
APPENDIX B2 *(STACY DALZELL)*

Processor Board Connectors Schematic



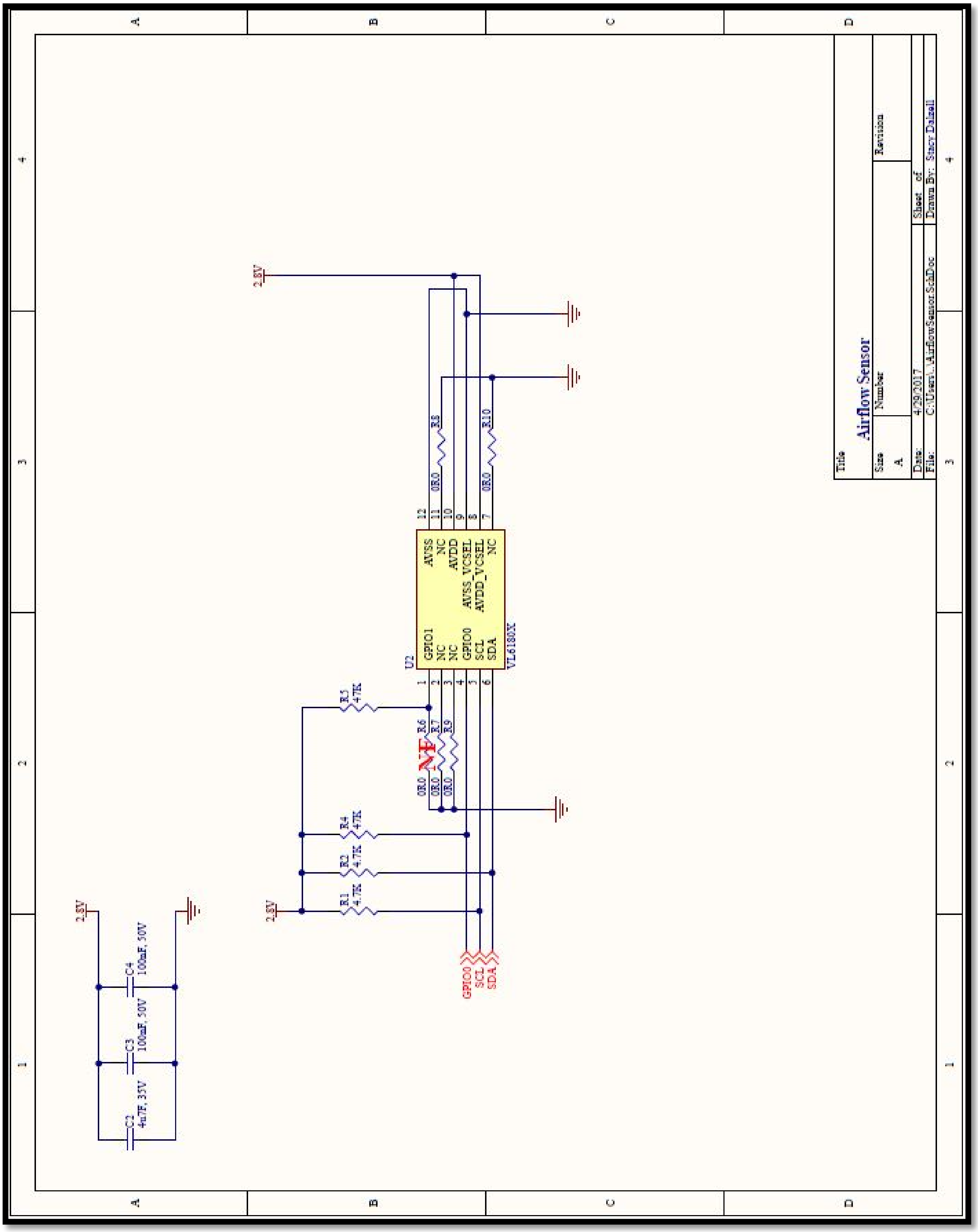
APPENDIX B3 *(STACY DALZELL)*

Airflow Sensor Board Connectors Schematic



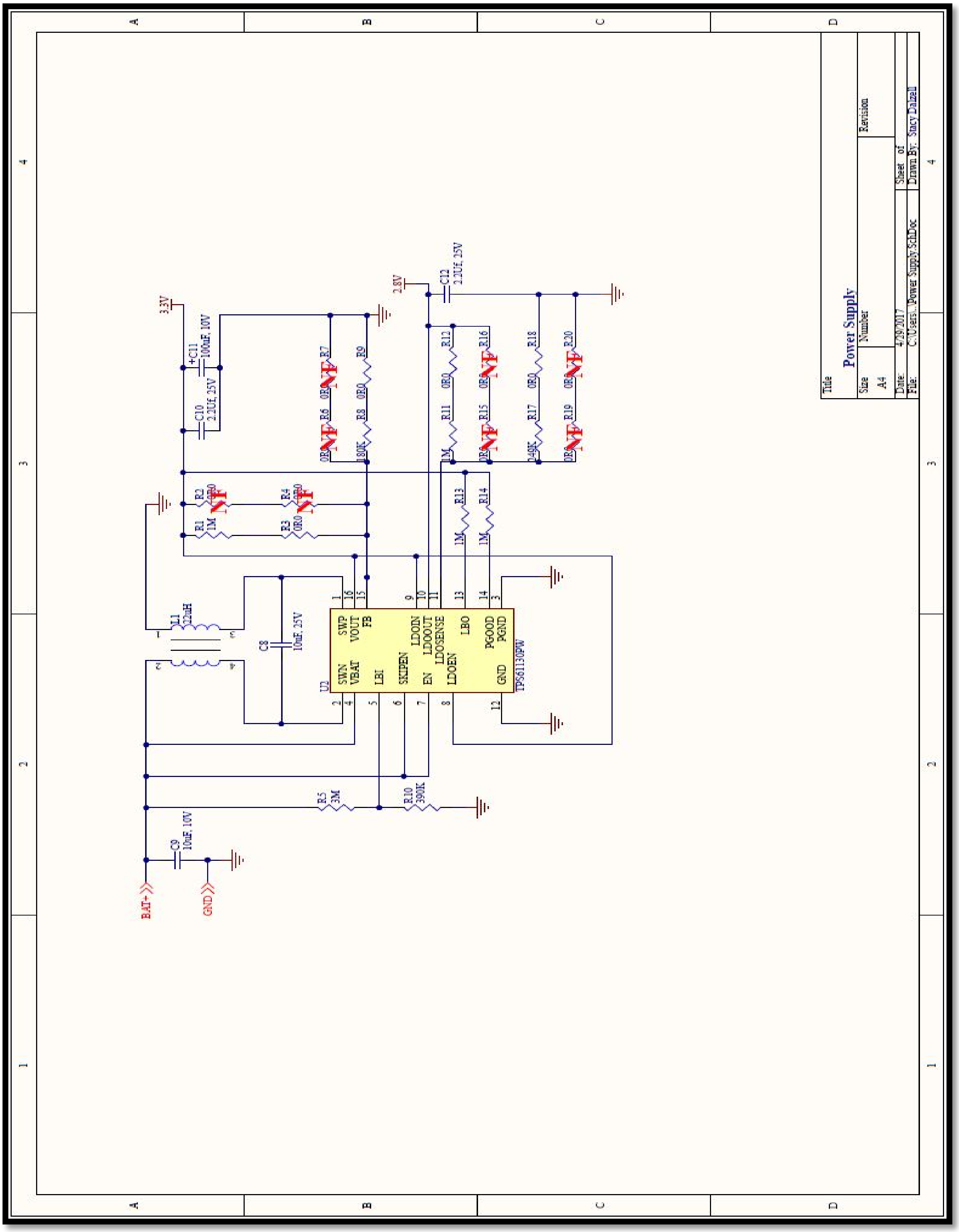
APPENDIX B4 *(STACY DALZELL)*

Air Flow Sensor Schematic



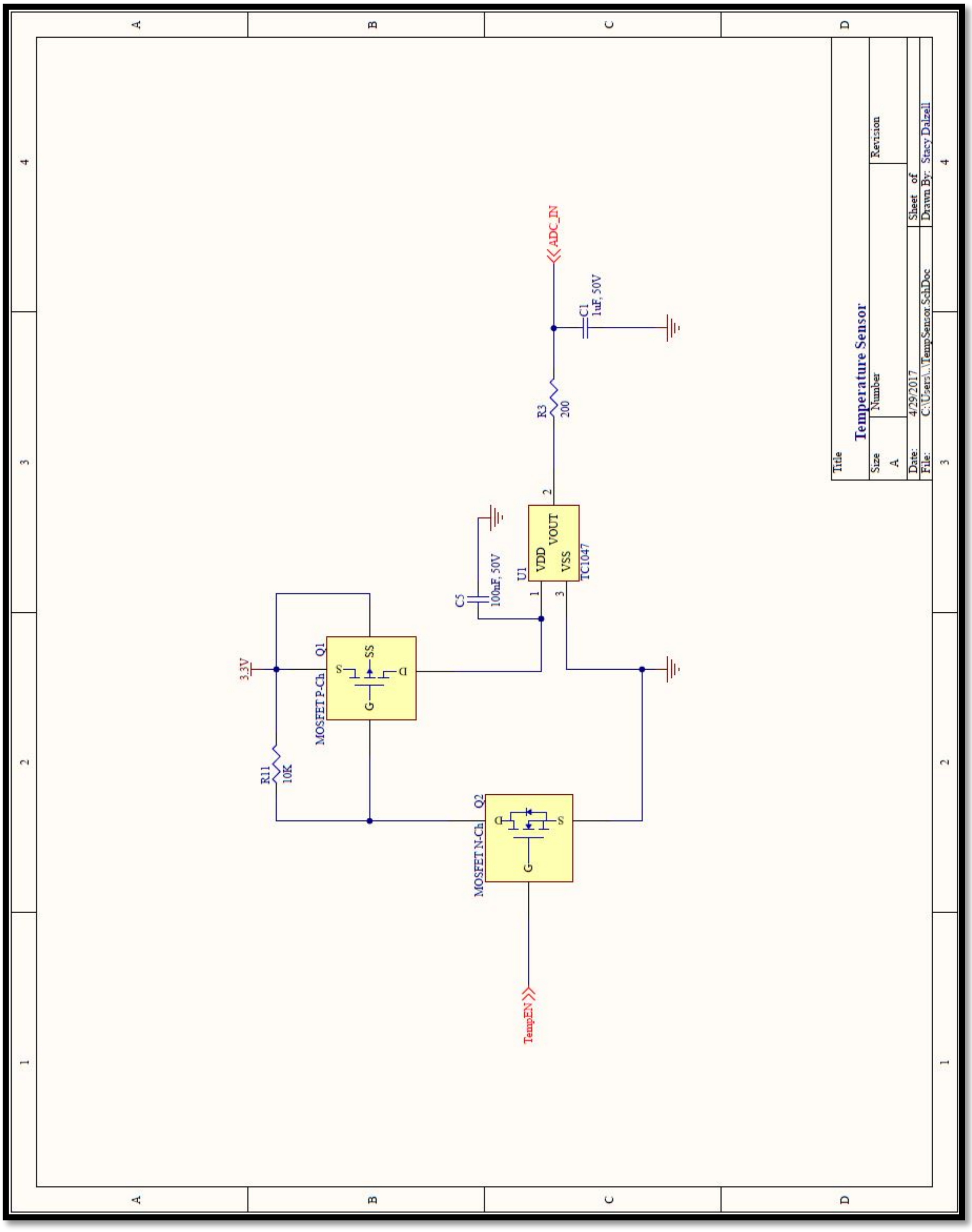
APPENDIX B5 *(STACY DALZELL)*

Power Supply Schematic



APPENDIX B6 *(STACY DALZELL)*

Temperature Sensor Schematic



APPENDIX C1 *(CONNOR MCCURLEY)*

This code was supplied by adafruit.com and was used to take measurements from the VL6180x IR distance sensor through utilization with an Arduino [9] [10] [11].

```
//--------------------------------------------------------------------------------
----------
//  Title:  Adafruit_VL6180X.ino
//  Author:  Adafruit Industries
//  Code Version:1.0
//  Availability:  http://learn.adafruit.com/adafruit-vl6180x-time-of-flight-micro-
lidar-distance-sensor-breakout/wiring-and-test
//--------------------------------------------------------------------------------
----------

#include <Wire.h>
#include "Adafruit_VL6180X.h"

Adafruit_VL6180X vl = Adafruit_VL6180X();

void setup() {
  Serial.begin(115200);

  // wait for serial port to open on native usb devices
  while (!Serial) {
    delay(1);
  }

  Serial.println("Adafruit VL6180x test!");
  if (! vl.begin()) {
    Serial.println("Failed to find sensor");
    while (1);
  }
  Serial.println("Sensor found!");
}

void loop() {
  float lux = vl.readLux(VL6180X_ALS_GAIN_5);

  Serial.print("Lux: "); Serial.println(lux);
  uint8_t range = vl.readRange();
  uint8_t status = vl.readRangeStatus();

  if (status == VL6180X_ERROR_NONE) {
    Serial.print("Range: "); Serial.println(range);
  }

  // Some error occurred, print it out!

  if  ((status >= VL6180X_ERROR_SYSERR_1) && (status <= VL6180X_ERROR_SYSERR_5)) {
    Serial.println("System error");
  }
  else if (status == VL6180X_ERROR_ECEFAIL) {
    Serial.println("ECE failure");
  }
  else if (status == VL6180X_ERROR_NOCONVERGE) {
    Serial.println("No convergence");
  }
  else if (status == VL6180X_ERROR_RANGEIGNORE) {
    Serial.println("Ignoring range");
  }
  else if (status == VL6180X_ERROR_SNR) {
    Serial.println("Signal/Noise error");
  }
  else if (status == VL6180X_ERROR_RAWUFLOW) {
    Serial.println("Raw reading underflow");
  }
  else if (status == VL6180X_ERROR_RAWOFLOW) {
    Serial.println("Raw reading overflow");
  }
  else if (status == VL6180X_ERROR_RANGEUFLOW) {
    Serial.println("Range reading underflow");
  }
  else if (status == VL6180X_ERROR_RANGEOFLOW) {
    Serial.println("Range reading overflow");
  }
  delay(50);
}
```

```
/**************************************************************************/
/*!
    @file     Adafruit_VL6180X.h
    @author   Limor Fried (Adafruit Industries)
      @license  BSD (see license.txt)

      This is a library for the Adafruit VL6180 ToF Sensor breakout board
      ----> http://www.adafruit.com/products/3316

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

      @section  HISTORY

    v1.0  - First release
*/
/**************************************************************************/

#include "Arduino.h"
#include <Wire.h>

//#define I2C_DEBUG

// the i2c address
#define VL6180X_DEFAULT_I2C_ADDR 0x29

#define VL6180X_REG_IDENTIFICATION_MODEL_ID    0x000
#define VL6180X_REG_SYSTEM_INTERRUPT_CONFIG    0x014
#define VL6180X_REG_SYSTEM_INTERRUPT_CLEAR     0x015
#define VL6180X_REG_SYSTEM_FRESH_OUT_OF_RESET  0x016
#define VL6180X_REG_SYSRANGE_START             0x018
#define VL6180X_REG_SYSALS_START               0x038
#define VL6180X_REG_SYSALS_ANALOGUE_GAIN       0x03F
#define VL6180X_REG_SYSALS_INTEGRATION_PERIOD_HI  0x040
#define VL6180X_REG_SYSALS_INTEGRATION_PERIOD_LO  0x041
#define VL6180X_REG_RESULT_ALS_VAL             0x050
#define VL6180X_REG_RESULT_RANGE_VAL           0x062
#define VL6180X_REG_RESULT_RANGE_STATUS        0x04d
#define VL6180X_REG_RESULT_INTERRUPT_STATUS_GPIO       0x04f

#define VL6180X_ALS_GAIN_1         0x06
#define VL6180X_ALS_GAIN_1_25      0x05
#define VL6180X_ALS_GAIN_1_67      0x04
#define VL6180X_ALS_GAIN_2_5       0x03
#define VL6180X_ALS_GAIN_5         0x02
#define VL6180X_ALS_GAIN_10        0x01
#define VL6180X_ALS_GAIN_20        0x00
#define VL6180X_ALS_GAIN_40        0x07

#define VL6180X_ERROR_NONE         0
#define VL6180X_ERROR_SYSERR_1     1
#define VL6180X_ERROR_SYSERR_5     5
#define VL6180X_ERROR_ECEFAIL      6
#define VL6180X_ERROR_NOCONVERGE   7
#define VL6180X_ERROR_RANGEIGNORE  8
```

```
#define VL6180X_ERROR_RAWUFLOW     12
#define VL6180X_ERROR_RAWOFLOW     13
#define VL6180X_ERROR_RANGEUFLOW   14
#define VL6180X_ERROR_RANGEOFLOW   15



class Adafruit_VL6180X {
 public:
  Adafruit_VL6180X();
  boolean begin(void);
  uint8_t readRange(void);
  float   readLux(uint8_t gain);
  uint8_t readRangeStatus(void);

 private:
  void loadSettings(void);

  void write8(uint16_t address, uint8_t data);
  void write16(uint16_t address, uint16_t data);

  uint16_t read16(uint16_t address);
  uint8_t read8(uint16_t address);

  uint8_t _i2caddr;
};
```

```
/**************************************************************************/
/*!
    @file     Adafruit_VL6180X.cpp
    @author   Limor Fried (Adafruit Industries)
      @license  BSD (see license.txt)

      This is a library for the Adafruit VL6180 ToF Sensor breakout board
      ----> http://www.adafruit.com/products/3316

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

      @section  HISTORY

    v1.0  - First release
*/
/**************************************************************************/

#include "Arduino.h"
#include <Wire.h>
#include "Adafruit_VL6180X.h"


/**************************************************************************/
/*!
    @brief  Instantiates a new VL6180X class
*/
/**************************************************************************/
Adafruit_VL6180X::Adafruit_VL6180X(void) {
}

/**************************************************************************/
/*!
    @brief  Setups the HW
*/
/**************************************************************************/
boolean Adafruit_VL6180X::begin(void) {
  _i2caddr = VL6180X_DEFAULT_I2C_ADDR;
  Wire.begin();

  if (read8(VL6180X_REG_IDENTIFICATION_MODEL_ID) != 0xB4) {
    return false;
  }

  //if (read8(VL6180X_REG_SYSTEM_FRESH_OUT_OF_RESET) == 0x01) {
    loadSettings();
  //}

  write8(VL6180X_REG_SYSTEM_FRESH_OUT_OF_RESET, 0x00);

  return true;
```

```
/**************************************************************************/
/*!
    @brief  Load the settings for ranging
*/
/**************************************************************************/

void Adafruit_VL6180X::loadSettings(void) {
    // load settings!

    // private settings from page 24 of app note
    write8(0x0207, 0x01);
    write8(0x0208, 0x01);
    write8(0x0096, 0x00);
    write8(0x0097, 0xfd);
    write8(0x00e3, 0x00);
    write8(0x00e4, 0x04);
    write8(0x00e5, 0x02);
    write8(0x00e6, 0x01);
    write8(0x00e7, 0x03);
    write8(0x00f5, 0x02);
    write8(0x00d9, 0x05);
    write8(0x00db, 0xce);
    write8(0x00dc, 0x03);
    write8(0x00dd, 0xf8);
    write8(0x009f, 0x00);
    write8(0x00a3, 0x3c);
    write8(0x00b7, 0x00);
    write8(0x00bb, 0x3c);
    write8(0x00b2, 0x09);
    write8(0x00ca, 0x09);
    write8(0x0198, 0x01);
    write8(0x01b0, 0x17);
    write8(0x01ad, 0x00);
    write8(0x00ff, 0x05);
    write8(0x0100, 0x05);
    write8(0x0199, 0x05);
    write8(0x01a6, 0x1b);
    write8(0x01ac, 0x3e);
    write8(0x01a7, 0x1f);
    write8(0x0030, 0x00);

    // Recommended : Public registers - See data sheet for more detail
    write8(0x0011, 0x10);       // Enables polling for 'New Sample ready'
                                // when measurement completes
    write8(0x010a, 0x30);       // Set the averaging sample period
                                // (compromise between lower noise and
                                // increased execution time)
    write8(0x003f, 0x46);       // Sets the light and dark gain (upper
                                // nibble). Dark gain should not be
                                // changed.
    write8(0x0031, 0xFF);       // sets the # of range measurements after
                                // which auto calibration of system is
                                // performed
    write8(0x0040, 0x63);       // Set ALS integration time to 100ms
    write8(0x002e, 0x01);       // perform a single temperature calibration
                                // of the ranging sensor

    // Optional: Public registers - See data sheet for more detail
    write8(0x001b, 0x09);       // Set default ranging inter-measurement
                                // period to 100ms
    write8(0x003e, 0x31);       // Set default ALS inter-measurement period
                                // to 500ms
    write8(0x0014, 0x24);       // Configures interrupt on 'New Sample
                                // Ready threshold event'
```

```
/**************************************************************************/
/*!
    @brief  Single shot ranging
*/
/**************************************************************************/

uint8_t Adafruit_VL6180X::readRange(void) {
  // wait for device to be ready for range measurement
  while (! (read8(VL6180X_REG_RESULT_RANGE_STATUS) & 0x01));

  // Start a range measurement
  write8(VL6180X_REG_SYSRANGE_START, 0x01);

  // Poll until bit 2 is set
  while (! (read8(VL6180X_REG_RESULT_INTERRUPT_STATUS_GPIO) & 0x04));

  // read range in mm
  uint8_t range = read8(VL6180X_REG_RESULT_RANGE_VAL);

  // clear interrupt
  write8(VL6180X_REG_SYSTEM_INTERRUPT_CLEAR, 0x07);

  return range;
}


/**************************************************************************/
/*!
    @brief  Error message (retreive after ranging)
*/
/**************************************************************************/

uint8_t Adafruit_VL6180X::readRangeStatus(void) {
  return (read8(VL6180X_REG_RESULT_RANGE_STATUS) >> 4);
}


/**************************************************************************/
/*!
    @brief  Single shot ranging
*/
/**************************************************************************/

float Adafruit_VL6180X::readLux(uint8_t gain) {
  uint8_t reg;

  reg = read8(VL6180X_REG_SYSTEM_INTERRUPT_CONFIG);
  reg &= ~0x38;
  reg |= (0x4 << 3); // IRQ on ALS ready
  write8(VL6180X_REG_SYSTEM_INTERRUPT_CONFIG, reg);

  // 100 ms integration period
  write8(VL6180X_REG_SYSALS_INTEGRATION_PERIOD_HI, 0);
  write8(VL6180X_REG_SYSALS_INTEGRATION_PERIOD_LO, 100);

  // analog gain
  if (gain > VL6180X_ALS_GAIN_40) {
    gain = VL6180X_ALS_GAIN_40;
  }
  write8(VL6180X_REG_SYSALS_ANALOGUE_GAIN, 0x40 | gain);

  // start ALS
  write8(VL6180X_REG_SYSALS_START, 0x1);
  // Poll until "New Sample Ready threshold event" is set
  while (4 != ((read8(VL6180X_REG_RESULT_INTERRUPT_STATUS_GPIO) >> 3) & 0x7));

  // read lux!
  float lux = read16(VL6180X_REG_RESULT_ALS_VAL);

  // clear interrupt
  write8(VL6180X_REG_SYSTEM_INTERRUPT_CLEAR, 0x07);

  lux *= 0.32; // calibrated count/lux
  switch(gain) {
  case VL6180X_ALS_GAIN_1:
    break;
  case VL6180X_ALS_GAIN_1_25:
    lux /= 1.25;
    break;
  case VL6180X_ALS_GAIN_1_67:
    lux /= 1.76;
    break;
  case VL6180X_ALS_GAIN_2_5:
    lux /= 2.5;
    break;
  case VL6180X_ALS_GAIN_5:
    lux /= 5;
    break;
  case VL6180X_ALS_GAIN_10:
    lux /= 10;
    break;
  case VL6180X_ALS_GAIN_20:
    lux /= 20;
    break;
  case VL6180X_ALS_GAIN_40:
    lux /= 20;
    break;
  }
  lux *= 100;
  lux /= 100; // integration time in ms


  return lux;
}

/**************************************************************************/
/*!
    @brief  I2C low level interfacing
*/
/**************************************************************************/


// Read 1 byte from the VL6180X at 'address'
uint8_t Adafruit_VL6180X::read8(uint16_t address)
{
  uint8_t data;

  Wire.beginTransmission(_i2caddr);
  Wire.write(address>>8);
  Wire.write(address);
  Wire.endTransmission();

  Wire.requestFrom(_i2caddr, (uint8_t)1);
```

```
#if defined(I2C_DEBUG)
  Serial.print("\t$"); Serial.print(address, HEX); Serial.print(": 0x"); Se-
rial.println(r, HEX);
#endif

  return r;
}


// Read 2 byte from the VL6180X at 'address'
uint16_t Adafruit_VL6180X::read16(uint16_t address)
{
  uint16_t data;

  Wire.beginTransmission(_i2caddr);
  Wire.write(address>>8);
  Wire.write(address);
  Wire.endTransmission();

  Wire.requestFrom(_i2caddr, (uint8_t)2);
  while(!Wire.available());
  data = Wire.read();
  data <<= 8;
  while(!Wire.available());
  data |= Wire.read();

  return data;
}

// write 1 byte
void Adafruit_VL6180X::write8(uint16_t address, uint8_t data)
{
  Wire.beginTransmission(_i2caddr);
  Wire.write(address>>8);
  Wire.write(address);
  Wire.write(data);
  Wire.endTransmission();

#if defined(I2C_DEBUG)
  Serial.print("\t$"); Serial.print(address, HEX); Serial.print(" = 0x"); Se-
rial.println(data, HEX);
#endif
}


// write 2 bytes
void Adafruit_VL6180X::write16(uint16_t address, uint16_t data)
{
  Wire.beginTransmission(_i2caddr);
  Wire.write(address>>8);
  Wire.write(address);
  Wire.write(data>>8);
  Wire.write(data);
  Wire.endTransmission();
}
```

APPENDIX C2 *(CONNOR MCCURLEY)*

Accuracy Test Code

This code was used to take measurements from the VL6180x IR distance sensor and a pitot-static tube sensor through utilization with an Arduino. The data measurements were logged to an SD card [9] [10] [11] [12].

```
//------------------------------------------------------------------------
------------------
//
//  Title:  Accuracy_Test.ino
//  Author: Connor McCurley
//  Date:  2017
//  Code Version: 1.0
//
//  Description:
//   This program is used to simultaneously measure the airflow readings
from a VL6108x IR
//  distance sensor and a pitot static sensor.  The distance measurement,
calculated airflow,
//   reference airflow, and difference between calculated and reference
airflow is recorded as
//  a .txt file on an SD card.
//
//  This program includes code borrowed from the following:
//
//  Title: Setra264.ino
//  Author: Stacy Dalzell
//  Date: 2017
//  Code Version: 1.0
//
```

```
//Include libraries

//Copied from Adafruit_VL6180X.ino ------------
#include <Wire.h>
#include "Adafruit_VL6180X.h"
//--------------------------------------------

// Copied from Setra264.ino
#include <LiquidCrystal.h>

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

//Create objects
File sensorData;

// Copied from Setra264.ino
Adafruit_VL6180X vl = Adafruit_VL6180X();

//Initialize variables
int data_mm;
float data_fpm;
float data_weighted;
float difference;

// Copied from Setra264.ino -----------------
int ADCin = 0;
float TempF = 70.0;
float Baro = 29.92;
float velocity = 0.0;
float ADCres = 0.004882813;
int PitotADC = 0;
float PitotPressure = 0;
float PitotRes = 0.5/1024;
float dryAirDensity = 1.325 * (Baro / (TempF + 460));
float velocity1 = 0.0;
float velocity2 = 0.0;
float velocity3 = 0.0;
float velocity4 = 0.0;
float velocity5 = 0.0;
float velocity6 = 0.0;
float velocity7 = 0.0;
float velocity8 = 0.0;
float velocity9 = 0.0;
float velocity10 = 0.0;
float velocity11 = 0.0;
float velocity12 = 0.0;
float velocity13 = 0.0;
float velocity14 = 0.0;
float velocity15 = 0.0;
float velocity16 = 0.0;
float velocity17 = 0.0;
float velocity18 = 0.0;
float velocity19 = 0.0;
float velocity20 = 0.0;
float velocity21 = 0.0;
float velocity22 = 0.0;
float velocity23 = 0.0;
float velocity24 = 0.0;
float velocity25 = 0.0;
float velocity26 = 0.0;
float velocity27 = 0.0;
float velocity28 = 0.0;
float velocity29 = 0.0;
float avgVelocity = 0.0;
//--------------------------------------------------

int enablePin = 8;
int tempEnable = 7;

//--------------------------------------------------------------------------------
-
void setup() {

//Enable VL6180x
pinMode(enablePin,OUTPUT);
digitalWrite(enablePin,HIGH);


//Test connection for SD card
if (!SD.begin(4)) {
    Serial.println("Initialization failed");
    return;
  } else{Serial.println("SD card initialization complete");}

  //Open SD data stream and write headers for data columns
  sensorData = SD.open("data.txt", FILE_WRITE);

  if(sensorData){
    //Serial.print("Able to write....");
    sensorData.println("Custom Sensor,Custom Weighted,Pitot Tube,Difference");
    sensorData.close();
  }

  else{ Serial.println("Error opening file");}

//Copied from Setra264.ino------------------
  //set pinMode for pitot tube reading
  pinMode(A1, INPUT);
  pinMode(A0, INPUT);
//------------------------------------------

  Serial.begin(9600);  //Begin Serial stream



while(!Serial){
  delay(1);
  }

//Copied from Adafruit_VL6180X.ino ------------
//Verify connection with Vl6180x
Serial.println("Beginning Program!");
if(vl.begin()){
  Serial.println("Failed to find sensor");
  }

  Serial.println("Sensor Found!");
//---------------------------------------------

  delay (500);
```

```
//--------------------------------------------------------------------------------
void loop() {

//Modified from Adafruit_VL6180X.ino ------------------------------------
//Get distance from Vl6180x and calculate expected airflow
uint8_t range = vl.readRange();
uint8_t status = vl.readRangeStatus();
if (status == VL6180X_ERROR_NONE) {
data_mm = range;
data_weighted = (2.0414*(data_mm*data_mm)) - (32.729*data_mm) - 375.37;
}
else{Serial.println("Error");}
//--------------------------------------------------------------------------

//Copied from Setra624.ino ----------------------------------------------
//Calculate pitot static airflow measurement
velocity29 = velocity28;
velocity28 = velocity27;
velocity27 = velocity26;
velocity26 = velocity25;
velocity25 = velocity24;
velocity24 = velocity23;
velocity23 = velocity22;
velocity22 = velocity21;
velocity21 = velocity20;
velocity20 = velocity19;
velocity19 = velocity18;
velocity18 = velocity17;
velocity17 = velocity16;
velocity16 = velocity15;
velocity15 = velocity14;
velocity14 = velocity13;
velocity13 = velocity12;
velocity12 = velocity11;
velocity11 = velocity10;
velocity10 = velocity9;
velocity9 = velocity8;
velocity8 = velocity7;
velocity7 = velocity6;
velocity6 = velocity5;
velocity5 = velocity4;
velocity4 = velocity3;
velocity3 = velocity2;
velocity2 = velocity1;
velocity1 = velocity;

ADCin = analogRead(A1);
PitotADC = analogRead(A1);
PitotPressure = PitotADC * PitotRes;
velocity = 1096.7 * sqrt(PitotPressure / dryAirDensity);
avgVelocity = (velocity29 + velocity28 + velocity27 + velocity26 + velocity25
                + velocity24 +  velocity23 + velocity22 + velocity21 + velocity20
                +velocity19 + velocity18 + velocity17 + velocity16 + velocity15
                + velocity14 +  velocity13 + velocity12 + velocity11 + velocity10
                + velocity9 + velocity8 + velocity7 + velocity6 + velocity5
                + velocity4 +  velocity3 + velocity2 + velocity1 + velocity) / 30;
//------------------------------------------------------------------------------------

data_fpm = avgVelocity;


//Calculate difference between calculated and reference airflow
difference = abs(data_fpm - data_weighted);
//Print data to serial log
//Serial.print("Custom: ");
//Serial.println(data_mm);
//Serial.print("weighted: ");
//Serial.println(data_weighted);
Serial.print("Pitot: ");
Serial.println(data_fpm);
//Serial.print("Difference: ");
//Serial.println(difference);

//Print data to SD card
sensorData = SD.open("data2.txt",FILE_WRITE);

    sensorData.print(data_mm);
    sensorData.print(",");

    sensorData.print(data_weighted);
    sensorData.print(",");

    sensorData.print(data_fpm);
    sensorData.print(",");

    sensorData.println(difference);
    sensorData.close();

delay(250);

}
```

APPENDIX D *(KEVIN HILL)*

```
/**
 ******************************************************************************
 * File Name          : main.c
 * Description        : Main program body
 ******************************************************************************
 * Edited by: Kevin Hill
              * Oklahoma State University
              * ECEN 4024 Spring 2017
 * COPYRIGHT(c) 2017 STMicroelectronics
 *
              * Included is the VL6180X library provided by Adafruit Industries
              * Author: Limor Fried (Adafruit Industries)
              * Date: 4/27/2017
              * Name: VL6180X Master Library
              * Code Version: v1.0
              * Type: Source Code
              * Address: https://learn.adafruit.com/adafruit-vl6180x-time-of-flight-micro-lidar-distance-sensor-breakout/wiring-and-test
 *
              * Low Level functions supplied by STMicroelectronics using STM32CubeMX Software
              *
              * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *   1. Redistributions of source code must retain the above copyright notice,
 *      this list of conditions and the following disclaimer.
 *   2. Redistributions in binary form must reproduce the above copyright notice,
 *      this list of conditions and the following disclaimer in the documentation
 *      and/or other materials provided with the distribution.
 *   3. Neither the name of STMicroelectronics nor the names of its contributors
 *      may be used to endorse or promote products derived from this software
 *      without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 ******************************************************************************
 */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32l0xx_hal.h"

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc;

I2C_HandleTypeDef hi2c1;

RTC_HandleTypeDef hrtc;

SPI_HandleTypeDef hspi1;

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/

#define VL6180X_DEFAULT_I2C_ADDR 0x29
```

```
#define VL6180X_REG_IDENTIFICATION_MODEL_ID    0x000
#define VL6180X_REG_SYSTEM_INTERRUPT_CONFIG    0x014
#define VL6180X_REG_SYSTEM_INTERRUPT_CLEAR     0x015
#define VL6180X_REG_SYSTEM_FRESH_OUT_OF_RESET  0x016
#define VL6180X_REG_SYSRANGE_START             0x018
#define VL6180X_REG_SYSALS_START               0x038
#define VL6180X_REG_SYSALS_ANALOGUE_GAIN       0x03F
#define VL6180X_REG_SYSALS_INTEGRATION_PERIOD_HI  0x040
#define VL6180X_REG_SYSALS_INTEGRATION_PERIOD_LO  0x041
#define VL6180X_REG_RESULT_ALS_VAL             0x050
#define VL6180X_REG_RESULT_RANGE_VAL           0x062
#define VL6180X_REG_RESULT_RANGE_STATUS        0x04d
#define VL6180X_REG_RESULT_INTERRUPT_STATUS_GPIO       0x04f

#define VL6180X_ALS_GAIN_1         0x06
#define VL6180X_ALS_GAIN_1_25      0x05
#define VL6180X_ALS_GAIN_1_67      0x04
#define VL6180X_ALS_GAIN_2_5       0x03
#define VL6180X_ALS_GAIN_5         0x02
#define VL6180X_ALS_GAIN_10        0x01
#define VL6180X_ALS_GAIN_20        0x00
#define VL6180X_ALS_GAIN_40        0x07

#define VL6180X_ERROR_NONE         0
#define VL6180X_ERROR_SYSERR_1     1
#define VL6180X_ERROR_SYSERR_5     5
#define VL6180X_ERROR_ECEFAIL      6
#define VL6180X_ERROR_NOCONVERGE   7
#define VL6180X_ERROR_RANGEIGNORE  8
#define VL6180X_ERROR_SNR          11
#define VL6180X_ERROR_RAWUFLOW     12
#define VL6180X_ERROR_RAWOFLOW     13
#define VL6180X_ERROR_RANGEUFLOW   14
#define VL6180X_ERROR_RANGEOFLOW   15
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void Error_Handler(void);
static void MX_GPIO_Init(void);
static void MX_ADC_Init(void);
static void MX_I2C1_Init(void);
static void MX_RTC_Init(void);
static void MX_SPI1_Init(void);
void Activate_Level_Shifter(void);
void Deactivate_Level_Shifter(void);
void Activate_Temp_Sensor(void);
void Deactivate_Temp_Sensor(void);
void Activate_SD(void);
void Deactivate_SD(void);
void Activate_Airflow_Sensor(void);
void Deactivate_Airflow_Sensor(void);
int i2cbegin(void);
void loadSettings(void);
uint8_t readRange(void);
uint8_t read8(uint16_t addr);
void write8(uint16_t,uint8_t);
int restartI2C(void);
void enterStop(uint32_t);
void printAirflow(uint8_t);
void printTemp(uint32_t);
double floor(double);
double ceil(double);


/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------*/
```

```
int main(void)
{
		uint32_t tempVal = 0;
		uint8_t airflowVal = 0;
		int c = 1;
		int d = 1;
  /* MCU Configuration----------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
		SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC_Init();
		MX_I2C1_Init();
  MX_RTC_Init();
  MX_SPI1_Init();

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
				//Read Temperature
				Activate_Temp_Sensor();

				//Delay for temp sensor to initialize
				for(c=1;c<= 50;c++)
				for(d = 1; d<50;d++)
				{}
				HAL_ADC_Start(&hadc);
				HAL_ADC_PollForConversion(&hadc,100000);
				tempVal = HAL_ADC_GetValue(&hadc);
				HAL_ADC_Stop(&hadc);

				//Read Airflow
				Activate_Level_Shifter();
				Activate_Airflow_Sensor();
				if(i2cbegin()==0){
				while(restartI2C()==0)
				{}}
				airflowVal = readRange();
				while(airflowVal == 0x04)
				{
					restartI2C();
					airflowVal=readRange();
				}

				//Deactivate All devices to save power.
				Deactivate_Temp_Sensor();
				Deactivate_Airflow_Sensor();
				Deactivate_Level_Shifter();

				//Print Statements
				uint8_t newLine = 0x0D;
				printAirflow(airflowVal);
				printTemp(tempVal);

				//Sleep for 30s
				enterStop(30000);
  }
}

void printTemp(uint32_t val)
{
		double newVal = val;
double temp = 0;

//Note if voltage reference for ADC changes, 3.3 must change
newVal= newVal*3.3;
newVal = newVal/4096;
newVal = newVal - 0.5;
newVal = newVal * 100;
if(newVal<1)
{
        temp=1;
}
int fracBits = 0;
uint8_t frac1 = 0x30;
uint8_t frac2 = 0x30;
uint8_t period = 0x2E;
if(newVal < (floor(newVal)+0.125))
{
        newVal = floor(newVal);
        fracBits = 0;
}
else if(newVal < (floor(newVal)+0.375))
{
        newVal = floor(newVal)+0.25;
        fracBits = 1;
}
else if(newVal < (floor(newVal)+0.625))
{
        newVal = floor(newVal)+0.5;
        fracBits = 2;
}
else if(newVal < (floor(newVal)+0.875))
{
        newVal = floor(newVal)+0.75;
        fracBits = 3;
}
else
{
        newVal = ceil(newVal);
        fracBits = 0;
}
//handles if the temp is <1, we do not need negative temperatures
if(temp==1)
{
        newVal=0;
}
uint8_t dec10 = newVal/10;
uint8_t dec1 = newVal-(10*dec10);

//Sets the decimal point in ASCII characters
if(fracBits==0)
{
        frac1 = 0x30;
        frac2 = 0x30;
}
else if (fracBits==1)
{
        frac1 = 0x32;
        frac2 = 0x35;
}
else if(fracBits == 2)
{
        frac1 = 0x35;
        frac2 = 0x30;
}
else if(fracBits == 2)
{
        frac1 = 0x37;
        frac2 = 0x35;
        }
        dec10 = dec10+0x30;
        dec1 = dec1+0x30;

        //Print Statements, print to SD will go here

        /*HAL_UART_Transmit(&huart2,&dec10,1,0xFFF);
        HAL_UART_Transmit(&huart2,&dec1,1,0xFFF);
        HAL_UART_Transmit(&huart2,&period,1,0xFFF);
        HAL_UART_Transmit(&huart2,&frac1,1,0xFFF);
        HAL_UART_Transmit(&huart2,&frac2,1,0xFFF);*/
}
void printAirflow(uint8_t val)
{
        uint16_t airflow = 0;
        double p1 = val;
        double p2 = val;
        double temp = 0;
        p1 = p1*p1;
        p1 = p1*2.0414;
        p2 = p2 * (-32.729);
        temp = p1 + p2;
        temp = temp - 375.37;
        if(temp<0)
        {temp = 0;}
        //Set 1000th,100th,10th, and 1st place
        uint8_t dec1000 = (airflow/0x3E8);
        uint8_t dec100 = (airflow%0x3E8)/0x64;
        uint8_t dec10 = ((airflow%0x3E8)%0x64)/0x0A;
        uint8_t dec1 = ((airflow%0x3E8)%0x64)%0x0A;

        //Move to ascii value
        dec1000 = dec1000+0x30;
        dec100 = dec100+0x30;
        dec10 = dec10+0x30;
        dec1 = dec1+0x30;

        //Print Statements will print to SD card here
        /*HAL_UART_Transmit(&huart2,&dec1000,1,0xFFF);
        HAL_UART_Transmit(&huart2,&dec100,1,0xFFF);
        HAL_UART_Transmit(&huart2,&dec10,1,0xFFF);
        HAL_UART_Transmit(&huart2,&dec1,1,0xFFF);*/
}
void enterStop(uint32_t ms)
{
        HAL_RTCEx_SetWakeUpTimer_IT(&hrtc,ms*2,RTC_WAKEUPCLOCK_RTCCLK_DIV16);
        HAL_PWR_EnterSTOPMode(PWR_MAINREGULATOR_ON,PWR_STOPENTRY_WFI);
        HAL_RTCEx_WakeUpTimerIRQHandler(&hrtc);
}
int restartI2C(void)
{
        Deactivate_Airflow_Sensor();
        Activate_Airflow_Sensor();
        MX_I2C1_Init();
        return i2cbegin();
}
uint8_t readRange(void)
{
        while(!(read8(VL6180X_REG_RESULT_RANGE_STATUS)&0x01))
        {}

        // Start a range measurement
        write8(VL6180X_REG_SYSRANGE_START,0x01);
  // Poll until bit 2 is set
  while (! (read8(VL6180X_REG_RESULT_INTERRUPT_STATUS_GPIO) & 0x04))
        {}
  // read range in mm
        uint8_t range = read8(VL6180X_REG_RESULT_RANGE_VAL);
  // clear interrupt
        write8(VL6180X_REG_SYSTEM_INTERRUPT_CLEAR,0x07);
  return range;
}
int i2cbegin(void)
{
        if(read8(VL6180X_REG_IDENTIFICATION_MODEL_ID)!= 0xB4)
        {
                return 0;
        }
  loadSettings();
        return 1;
}
uint8_t read8(uint16_t addr)
{
        uint8_t data = 0;
        uint8_t firstbits = addr>>8;
        uint8_t lastbits = addr;
        uint8_t outbuffer[2] = {firstbits,lastbits};
        HAL_I2C_Master_Transmit(&hi2c1,0x52,outbuffer,sizeof(outbuffer),0xFFF);
        HAL_I2C_Master_Receive(&hi2c1,0x53,&data,1,0xFFF);//problem
        return data;
}
void write8(uint16_t addr,uint8_t data)
{
        uint8_t firstbits = addr>>8;//transmit first 8 bits
        uint8_t lastbits = addr;//transmit last 8 bits
        uint8_t outbuffer[3] = {firstbits,lastbits,data};
        HAL_I2C_Master_Transmit(&hi2c1,0x52,outbuffer,sizeof(outbuffer),0xFFF);
}
void loadSettings(void)
{
                // load settings!
   // private settings from page 24 of app note
   write8(0x0207, 0x01);
   write8(0x0208, 0x01);
   write8(0x0096, 0x00);
   write8(0x0097, 0xfd);
   write8(0x00e3, 0x00);
   write8(0x00e4, 0x04);
   write8(0x00e5, 0x02);
   write8(0x00e6, 0x01);
   write8(0x00e7, 0x03);
   write8(0x00f5, 0x02);
   write8(0x00d9, 0x05);
   write8(0x00db, 0xce);
   write8(0x00dc, 0x03);
   write8(0x00dd, 0xf8);
   write8(0x009f, 0x00);
   write8(0x00a3, 0x3c);
   write8(0x00b7, 0x00);
   write8(0x00bb, 0x3c);
   write8(0x00b2, 0x09);
   write8(0x00ca, 0x09);
   write8(0x0198, 0x01);
   write8(0x01b0, 0x17);
   write8(0x01ad, 0x00);
   write8(0x00ff, 0x05);
   write8(0x0100, 0x05);
   write8(0x0199, 0x05);
   write8(0x01a6, 0x1b);
   write8(0x01ac, 0x3e);
   write8(0x01a7, 0x1f);
   write8(0x0030, 0x00);

   // Recommended : Public registers - See data sheet for more detail
   write8(0x0011, 0x10);       // Enables polling for 'New Sample ready'
                              // when measurement completes
   write8(0x010a, 0x30);      // Set the averaging sample period
                              // (compromise between lower noise and
                              // increased execution time)
   write8(0x003f, 0x46);      // Sets the light and dark gain (upper
                              // nibble). Dark gain should not be
                              // changed.
   write8(0x0031, 0xFF);      // sets the # of range measurements after
                              // which auto calibration of system is
                              // performed
   write8(0x0040, 0x63);      // Set ALS integration time to 100ms
   write8(0x002e, 0x01);      // perform a single temperature calibration
                              // of the ranging sensor

   // Optional: Public registers - See data sheet for more detail
   write8(0x001b, 0x09);      // Set default ranging inter-measurement
                              // period to 100ms
   write8(0x003e, 0x31);      // Set default ALS inter-measurement period
                              // to 500ms
   write8(0x0014, 0x24);      // Configures interrupt on 'New Sample
                              // Ready threshold event'
}
void Activate_Airflow_Sensor(void)
{
        HAL_GPIO_WritePin(GPIOB,GPIO_PIN_5, GPIO_PIN_SET);
}

void Deactivate_Airflow_Sensor(void)
{
        HAL_GPIO_WritePin(GPIOB,GPIO_PIN_5, GPIO_PIN_RESET);
}
void Activate_SD(void)
{
        HAL_GPIO_WritePin(GPIOB,GPIO_PIN_0, GPIO_PIN_SET);
}

void Deactivate_SD(void)
{
        HAL_GPIO_WritePin(GPIOB,GPIO_PIN_0, GPIO_PIN_RESET);
}
void Activate_Temp_Sensor(void)
{
        HAL_GPIO_WritePin(GPIOB,GPIO_PIN_3, GPIO_PIN_SET);
}

void Deactivate_Temp_Sensor(void)
{
        HAL_GPIO_WritePin(GPIOB,GPIO_PIN_3, GPIO_PIN_RESET);
}
void Activate_Level_Shifter(void)
{
        HAL_GPIO_WritePin(GPIOB,GPIO_PIN_1, GPIO_PIN_SET);
}

void Deactivate_Level_Shifter(void)
{
        HAL_GPIO_WritePin(GPIOB,GPIO_PIN_1, GPIO_PIN_RESET);
}
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_PeriphCLKInitTypeDef PeriphClkInit;

   /**Configure the main internal regulator output voltage
   */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
```

```
    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_LSI|RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.LSIState = RCC_LSI_ON;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_5;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }

  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_I2C1|RCC_PERIPHCLK_RTC;
  PeriphClkInit.I2c1ClockSelection = RCC_I2C1CLKSOURCE_PCLK1;
  PeriphClkInit.RTCClockSelection = RCC_RTCCLKSOURCE_LSI;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }

  HAL_RCC_MCOConfig(RCC_MCO1, RCC_MCO1SOURCE_SYSCLK, RCC_MCODIV_1);

    /**Configure the Systick interrupt time
    */
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

    /**Configure the Systick
    */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* ADC init function */
static void MX_ADC_Init(void)
{

  ADC_ChannelConfTypeDef sConfig;

    /**Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)
    */
  hadc.Instance = ADC1;
  hadc.Init.OversamplingMode = DISABLE;
  hadc.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV1;
  hadc.Init.Resolution = ADC_RESOLUTION_12B;
  hadc.Init.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  hadc.Init.ScanConvMode = ADC_SCAN_DIRECTION_FORWARD;
  hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc.Init.ContinuousConvMode = DISABLE;
  hadc.Init.DiscontinuousConvMode = DISABLE;
  hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc.Init.DMAContinuousRequests = DISABLE;
  hadc.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  hadc.Init.Overrun = ADC_OVR_DATA_PRESERVED;
  hadc.Init.LowPowerAutoWait = DISABLE;
  hadc.Init.LowPowerFrequencyMode = DISABLE;
  hadc.Init.LowPowerAutoPowerOff = DISABLE;
  if (HAL_ADC_Init(&hadc) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure for the selected ADC regular channel to be converted.
    */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
  if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

}

/* I2C1 init function */
static void MX_I2C1_Init(void)
{

  hi2c1.Instance = I2C1;
  hi2c1.Init.Timing = 0x00000708;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure Analogue filter
    */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
    Error_Handler();
  }

}

/* RTC init function */
static void MX_RTC_Init(void)
{

    /**Initialize RTC Only
    */
  hrtc.Instance = RTC;
  hrtc.Init.HourFormat = RTC_HOURFORMAT_24;
  hrtc.Init.AsynchPrediv = 127;
  hrtc.Init.SynchPrediv = 255;
  hrtc.Init.OutPut = RTC_OUTPUT_DISABLE;
  hrtc.Init.OutPutRemap = RTC_OUTPUT_REMAP_NONE;
  hrtc.Init.OutPutPolarity = RTC_OUTPUT_POLARITY_HIGH;
  hrtc.Init.OutPutType = RTC_OUTPUT_TYPE_OPENDRAIN;
  if (HAL_RTC_Init(&hrtc) != HAL_OK)
  {
    Error_Handler();
  }
```

```
}

/* SPI1 init function */
static void MX_SPI1_Init(void)
{

  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 7;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }

}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
     PA8   ------> RCC_MCO
*/
static void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_3|GPIO_PIN_5, GPIO_PIN_RESET);

  /*Configure GPIO pins : PB0 PB1 PB3 PB5 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_3|GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pin : PA8 */
  GPIO_InitStruct.Pin = GPIO_PIN_8;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  GPIO_InitStruct.Alternate = GPIO_AF0_MCO;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : PA15 */
  GPIO_InitStruct.Pin = GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

}
```

```
/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler */
  /* User can add his own implementation to report the HAL error return state */
  while(1)
  {
  }
  /* USER CODE END Error_Handler */
}

#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */

}

#endif

/**
  * @}
  */

/**
  * @}
*/

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

APPENDIX E *(STACY DALZELL)*

Project Specifications

System Overview

The device will autonomously measure and record temperature and airflow in a ventilation duct. The device must be able to be placed around the first bend in a four-inch duct without disassembling the duct. The installation of the device will not result in a significant restriction in airflow. The device will be expected to log data for a minimum of 30 days.

Physical Characteristics

Maximum Processor/Battery Enclosure Size (L x W x H): 3 in. x 4 in. x 1.5 in.
Operating Temperature: 4-55 °C

Electrical Characteristics

<u>*Battery Characteristics*</u>

Average Current Consumption: < 2.5 mA
Battery Capacity: 2000 mAh
Battery Life: ≥ 30 days
Internal Battery Discharge: < 20% per month

Battery capacity is rated by the mAh of current produced by the battery between a full charge of approximately 4.2 V and a defined discharge state of 3.3 V under a constant current load. To characterize the capacity of the battery a constant current load of 250 mA is provided by a Kikusui Model PLZ 72W electronic load and the time for the battery voltage to drop to 3.3 V is recorded. Battery capacity is calculated with the formula below.

$$mAh = \frac{Constant\ Current\ Load\ (mA)}{\#\ hours}$$

Battery life is calculated with the formula below.

$$Battery\ Life\ (days) = \frac{Battery\ Capacity\ (mAh)}{Average\ Current\ (mA) \times 24 + Internal\ Discharge\ Current}$$

Voltage Regulation

The Lithium battery will be input to a buck - boost voltage regulator. The specific voltage regulator is TPS6113xPW, made by Texas Instruments. The voltage regulator will step down the battery voltage and produce a constant output voltage of 3.3 V. The input current from the battery is a function of input voltage for the regulator. The specific component specifications are shown below.

Operating Temperature: -40 °C to 85 °C	Output Current: 320 mA
Operating Supply Current: 40 µA	Output Voltage: 3.3 V
Input Voltage: 3 - 4 V	

The specifications of the voltage regulator will be tested using the following steps:

1. Various input voltages ranging from 3 V to 4 V (operating range voltage) will be introduced to the TPS6113xPW circuit.
2. The resulting output voltage will be recorded. Output current and drop out voltage will also be recorded.
3. The collected data will be used to determine if the TPS6113xPW meets project specifications and power requirements.

Temperature Sensor Characteristics

<u>*Function*</u>

The Microchip TC1047 based temperature sensor measures the air temperature inside the ventilation duct with a response time of less than 30 seconds. The response time is based on the time it takes to perform the temperature measurement cycle.
The temperature measurement cycle is defined below.

1. The temperature sensor exits a low power state.
2. The measurement is performed.
3. The data conversion is performed.
4. The data is stored.

5. The sensor returns to a low power state.

Interface

The temperature sensor produces an analog output voltage in the range of 0.4 V to 1.1 V. The temperature relationship to the output voltage is

$$Temperature\ (℃) = \frac{Output\ Voltage\ (mV) - 500\ mV}{10\ mV}$$

Accuracy

Temperature sensor accuracy: ±0.5°C @ 0 °C
Accuracy is measured by immersing the sensor in a distilled ice and water mix. The ADC conversion accuracy will be less than 4 LSB (1 LSB = 8.0586 x 10^{-4} V with a 3.3 V ADC voltage reference). This will be tested by inputting a DC voltage into the ADC and comparing this value to the output of the ADC.

Airflow Sensor Characteristics

Sensor Model

A VL6180X time-of-flight distance sensor will be used to measure the position of a rotating vane within the air duct. The readings of the vane's position will be scaled appropriately to demonstrate the air speed inside the duct.

Function

The airflow sensor will measure wind speeds inside the ventilation duct with a minimum airflow accuracy of: ±15 feet per minute with speeds less than 150 feet per minute, and ±10% at speeds ranging from 150-1500 feet per minute.

The sensor will be capable of waking from sleep, taking a measurement, and powering down in under 30 seconds.

Interface

The time-of-flight distance sensor operates on 2.8V and is able to be sourced from 3-5V. It will transmit binary data to the microcontroller using an 8-bit I2C interface.

Accuracy

The sensor's proximity ranging is specified to be accurate to the millimeter between 0 to 100mm (3.937in), at 23 °C. Range offset error accumulates to 13mm (0.5in) when attempting to read beyond a distance of 100mm (3.937in).

Distance accuracy will first be measured by locating an object placed at discrete distances away from the sensor, between 0 and 100mm. After distance-measuring accuracy has been determined, scaling values will be selected in reference to the vane's rotation, which will symbolize air speed in the duct. This set of values, obtained from the scaling, will be compared to the air speed calculated using a differential pressure measurement taken from a water manometer.

Processor and Data Storage Characteristics

Function

The processor and data storage subsystem must take data gathered by the sensors and store it to a micro SD card in a way that is easy for a non-technical end user to read and access.

Interfaces

Sensor reading – Data incoming from the temperature sensor will be an analog voltage. This must then be converted to a digital value using the Cortex M0's built in 12-bit analog to digital converter.
Data from the airflow sensor will be an 8-bit digital signal.

Data Writing – Data will be written to the SD card using the serial peripheral interface(SPI). The file type will be a .txt file, with sensor data and a timestamp on each line.

Batch Writing - To conserve power, sensor data will be stored locally on the processor for a certain amount of time. Then, the processor will write the locally stored data to the SD card. This will require an interface such as a button to press for the user to write all unwritten data to the SD card before it is ejected.

Usability

Battery replacement and data recovery to be accomplished by a non-technical person with simple hand tools.

Revision History

Revision	Date	Approved By	Signature	Remarks
1	1/31/2017	Christian Bach, Ph.D.	N/A	Preliminary Draft
2	3/1/2017	Christian Bach, Ph.D.	N/A	Second Draft
3	3/7/2017	Christian Bach, Ph.D.		Final
		Stacy Dalzell		
		Kevin Hill		
		Aaisha Khalid		
		Connor McCurley		

APPENDIX F *(STACY DALZELL)*

Project Budget Table

Sub-System	Component	Unit Cost	Qty.	Total
Power	Battery	$13.00	1	$13.00
	Voltage Regulator	$0.80	2	$1.60
	Power Management	$1.00	1	$1.00
Processor & Data Storage	Cortex M0	$3.00	1	$3.00
	Micro SD Card	$15.00	1	$15.00
	Micro SD Card Holder	$3.50	1	$3.50
Temperature Sensor	STLM20 Sensor	$0.81	1	$0.81
	TC1047 Sensor	$0.62	1	$0.62
	DS18B20 Sensor	$3.00	1	$3.00
Airflow Sensor	HC-SR04 Ultrasonic Sensor	$3.00	2	$6.00
	3" PVC	$6.89	1	$6.89
PCB	Processor PCB (4 layer - $/sq. in.)	$10.00	4	$40.00
	Sensor PCB (2 layer - $/sq. in.)	$5.00	4	$20.00
Miscellaneous	Passive Components	$20.00		$20.00
	Connectors	$20.00		$20.00
	Shipping	$30.00		$30.00
			Total	$184.42

APPENDIX G *(AAISHA KHALID)*

Project Advisor Meeting Minutes

Project Advisor Meeting			
Date	**Start Time**	**End Time**	**Total Meeting Minutes**
1/26/2017	11:19AM	11:53AM	34
2/2/2017	11:05AM	11:36AM	31
2/9/2017	12:00PM	12:10PM	10
2/16/2017	11:06AM	11:26AM	20
2/23/2017	10:47AM	11:00AM	13
3/2/2017	11:02AM	11:17AM	15
3/23/2017	10:57AM	11:19AM	22
3/30/2017	10:55AM	11:10AM	15
4/13/2017	11:01AM	11:15AM	14
4/27/2017	2:15PM	2:45PM	30
	Overall Meeting Minutes		204

APPENDIX H *(AAISHA KHALID)*

Project Team Meeting Minutes

Project Team Meeting			
Date	**Start Time**	**End Time**	**Total Meeting Minutes**
1/24/2017	11:00AM	2:15PM	195
1/26/2017	12:00PM	12:22PM	22
1/28/2017	6:00PM	7:00PM	60
1/30/2017	3:00PM	4:00PM	60
2/2/2017	11:50AM	12:45PM	55
2/9/2017	12:20PM	1:30PM	70
3/2/2017	11:30AM	12:47PM	77
3/9/2017	1:00PM	2:00PM	60
	Overall Meeting Minutes		599

REFERENCES *(AAISHA KHALID)*

Works Cited

[1] STMicroelectronics, "VL6180X Revision 7," STMicroelectronics, 2016. [Online]. [Accessed 30 January 2017].

[2] Microchip, "TC1047/TC1047A, Precision Temperature-to-Voltage Converter datasheet," Microchip, 1 March 2012. [Online]. [Accessed 19 January 2017].

[3] T. Instruments, "TPS6113x Synchronous SEPIC and Flyback Converter With 1.1-A Switch and Integrated LDO," Texas Instruments, 2016. [Online]. [Accessed 30 January 2017].

[4] U. D. o. Energy, "Support at intervals for flex ducts," US Department of Energy, January 2017. [Online]. Available: https://basc.pnnl.gov/resource-guides/support-intervals-flex-ducts. [Accessed 28 March 2017].

[5] T. Instruments, "1-A High Efficiency Step-down Buck Converter in SOT23 and SOT563 Package," Texas Instruments, 2017. [Online]. Available: http://www.ti.com/product/TLV62568/datasheet. [Accessed 15 January 2017].

[6] P. Stream, "How to Calculate Battery Run-Time," PowerStream Technology, 3 Febuary 2017. [Online]. Available: http://www.powerstream.com/battery-capacity-calculations.htm. [Accessed 29 March 2017].

[7] STMicroelectronics, "STM32CubeMX," STMicroelectronics, 2017. [Online]. [Accessed 30 January 2017].

[8] F. Grieser, "pla vs. abs: 3d printer filaments compared," 16 January 2016. [Online]. Available: https://all3dp.com/pla-abs-3d-printer-filaments-compared/. [Accessed 3 March 2017].

[9] L. Fried, "Adafruit_VL6180X.cpp (Version 1.0)[Source Code]," Adafruit Industries, 2017. [Online]. Available: https://learn.adafruit.com/adafruit-vl6180x-time-of-flight-micro-lidar-distance-sensor-breakout/wiring-and-test. [Accessed 19 February 2017].

[10] L. Fried, "Adafruit_VL6180X.h (Version 1.0)[Source Code]," Adafruit Industries , 2017. [Online]. Available: https://learn.adafruit.com/adafruit-vl6180x-time-of-flight-micro-lidar-distance-sensor-breakout/wiring-and-test. [Accessed 17 February 2017].

[11] A. Industries, " Adafruit_VL6180X.ino (Version 1.0)[Source Code]," Adafruit Industries, 2017. [Online]. Available: https://learn.adafruit.com/adafruit-vl6180x-time-of-flight-micro-lidar-distance-sensor-breakout/wiring-and-test. [Accessed 15 February 2017].

[12] S. Dalzell, *Setra264.ino (Version 1.0)[Source Code],* 2017.

[13] Sparkfun, "Lithium Ion Battery - 2000mAh," Sparkfun, 2017. [Online]. Available: https://www.sparkfun.com/products/8483. [Accessed 20 January 2017].

[14] Wikipedia, "Oklahoma State University - Stillwater," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Oklahoma_State_University%E2%80%93Stillwater. [Accessed 30 April 2017].

[15] Wikipedia, "Pistol Pete (Oklahoma State University)," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Pistol_Pete_(Oklahoma_State_University). [Accessed 30 April 2017].

[16] O. Foundation, "Orange Passion: Engineering, Architecture & Technology," 2017. [Online]. Available: https://osugiving.com/yourpassion/college-of-engineering-architecture-and-technology. [Accessed 30 April 2017].