Spring 2016

HVAC Monitoring System

Ryan Gerhart
rmg44@zips.uakron.edu

Tyler Miller
tam69@zips.uakron.edu

Ryan Gerhart -

For the design process, I was the team leader, but more specifically in charge of the data collection aspect of the system. This means I was in charge of obtaining accurate data in the system and developing the ability to have that data sent out to be stored in a developed GUI for observation. For our sensors, I had to find an airflow, humidity, and temperature sensor that would be able to track the data with precision, while staying inside our limited budget. We settled on a temperature sensor that was placed below the energy harvesting Savonious wind turbine that would not be hindered by any other system in how it obtained data. This temperature sensor also came with a built-in humidity sensor, which will change in the system based on the temperature that the HVAC unit is set at. For airflow, I decided on a differential pressure sensor, which would monitor the airflow as the difference in airflow between the two sensing nodes. In order to obtain accurate data for observation, the sensor was placed next to the temperature sensor, oriented in such a way that would allow for an accurate measurement. Since the sensor measured differential pressure, I had to use a formula to translate this differential pressure in Pascals into an airflow data point in CFM, based in part on the size of the duct. When we finalized the report, I was tasked with combining the sections that each team member had written and making all the final changes regarding layout, punctuation, grammar, and minor mark-ups from advisors.

Tyler Allen Miller -

During the senior design process, I was given the task of hardware manager. More specifically, I was in charge of power, both satisfying power needs of internal systems necessary to perform all tasks, as well as keeping the system safe during the charging process. During the design process, I had to provide a size-limited power source that would satisfy all necessary power requirements, while being able to be confined inside the autonomous system. Also, it was necessary for the system to recharge this power source and take into consideration a potential overcharge or an input power over safe charge. This came to fruition when I selected a lithium-ion polymer rechargeable battery and backpack unit. This unit would provide the battery with a safe voltage regulation in case of an incoming voltage above a safe charging value. Lastly, in order to keep the flow of power linear, I was tasked with specifying a diode that would be able to safely charge the battery while not overheating. The diode I chose was well beyond any power that was possible to create in our HVAC monitoring device (600V and 1A respectively) Finally, while a large majority of my designs and implementations were at the end of the design process, I spent a large amount of time updating our design report during the semester to satisfy any change in graphics, tables and specifications throughout the design process.

HVAC Monitoring Device

Final Design and Implementation Report

Design Team I

Ryan Gerhart

TJ Ghinder

Tyler Miller

Nicholas Owens

Dr. J. Alexis De Abreu-Garcia

April 22, 2016

Table of Contents

List of Figures

List of Tables

Abstract: *This report outlines the design and implementation stages of the development of a heating, ventilation, and air conditioning (HVAC) airflow, humidity, and temperature monitoring device. The purpose of the device is to collect data to inform manufacturers and users about the current operational status of the HVAC system in place. Information about the airflow, humidity, and temperature is to be collected and wirelessly transmitted to a central database, where the data can be stored for later use and processing. The sensors and transmitter are to be powered using only energy harvested from the system, so that this data can be obtained at no additional cost to the consumer.*

1. Problem Statement

1.1 Need [TG]

HVAC systems are ubiquitous in modern day architecture. The U.S. Energy Information Agency published data this past year which states that heating and cooling accounts for 48 percent of the energy used in a typical office or home [10]. As global temperatures continue to rise, reducing the carbon footprint of humanity has become a primary concern of system designers and homeowners alike.

1.2 Objective [TG]

By introducing a device which can actively monitor the operational status of an HVAC unit from its source, problems with the unit may be detected sooner and therefore resolved faster, at minimal cost to the consumer. By using only energy provided by the airstream, this device will reduce the overall carbon footprint of the system at hand, as well as improve system reliability and longevity for the consumer.

The objective of the HVAC Airflow Monitoring device is to provide accurate data about the operation of the HVAC system. In order for the device to be cost-effective, it must use power generated only through the airflow of the system, but must not interrupt the airstream in a way that would be intrusive to the standard operation of the system.

This HVAC Airflow Monitoring device is intended to be a small part of a larger system designed to help curb the energy footprint of the HVAC unit. By feeding back information about the HVAC current operational status, problems may be detected sooner, and the overall effectiveness of the system can be improved. In addition, if enough energy is generated, it could be possible to

feed some of that energy back into the system to help power the compressor motor or blower. The latter however is outside the scope of this project.

1.3 Background

1.3.1 Airflow Monitoring [RG, TJ]

The current standard detecting method for an HVAC airflow uses a thermistor and a hot wire with a control circuit, referred to as a Mass Airflow Sensor, [1][2]. A thermistor is used to measure the temperature of the air in the system while a hot wire is kept at a constant temperature in relation to the thermistor by the control circuit. As airflow increases the wire's temperature decreases, prompting the control circuit to compensate by increasing the current through the wire. The control circuit measures the amount of current it has to output and relates that to a voltage signal that can be processed by a computer to determine the airflow, [1][2].

EnOcean Inc. has created a sensor, which is powered by the temperature differential between the inside and outside of an HVAC duct, to wirelessly transmit data about the temperature and airflow at the location of the sensor. It generates power using either an ECT 310 converter or a solar cell. The transmission of information is done using either a whip or helical antenna, and it operates at 868 MHz (Europe) or 315 MHz (in the USA, Japan, and China). The patent filed is for an "ultra-low-voltage DC/DC converter which enables low-cost standard Peltier elements to power the battery-less EnOcean wireless modules using wasted thermal energy."

Each of these types of technologies has been designed explicitly for use within a duct; this differs from our design in that our HVAC monitoring device is intended to be placed at the duct entrance to the system, to monitor the operational status of the HVAC unit itself. Currently, there is no technology which has been developed with the express purpose of providing information about the HVAC unit's operational status using only energy from the unit's airstream. This is usually done using status sensors within the unit (which consume large amounts of power), or by a maintenance worker after a problem has occurred, [6].

1.3.2 Wireless Transmission of Data [NO]

ZigBee-based wireless networks provide low power, low cost wireless networks that allow data to be sent from standalone sensors to a primary node. Some ZigBee devices contain power management systems that reduce their power requirements. The ZigBee protocol and software are open source and thus free to users. Data rates can vary from 20kbit/s to 250kbit/s, [3].

Picoradio networks are geared toward a sensor-based environment. Sensor data rates are quite low and sensors do not have to be awake all the time, allowing the network to be low power, [4]. Similar to ZigBee, Picoradio is an Ad-Hoc Network, allowing a low

cost, expansive network. However, compared to ZigBee, there is not a large source of software and protocols readily available to users.

1.3.1 Sensor Background [RG]

A method for detecting an HVAC airflow, referred to as a Mass AirFlow Sensor, uses a thermistor, a hot wire, and a control circuit. The thermistor is used to measure the temperature of the air in the system. The hot wire is kept at a constant temperature in relation to the thermistor by the control circuit. As airflow increases the wire's temperature decreases, prompting the control circuit to compensate by increasing the current through the wire. The control circuit measures the amount of power it has to output and relates that to a voltage signal that can be processed by a computer to determine the airflow. Since this procedure is heavily used in numerous applications, ranging from automotive to HVAC systems, this option was researched thoroughly, as explained in the Airflow Sensor part of the Technical Design. An example circuit of a Hot Wire Mass AirFlow Sensor can be seen below in **Figure 1**.

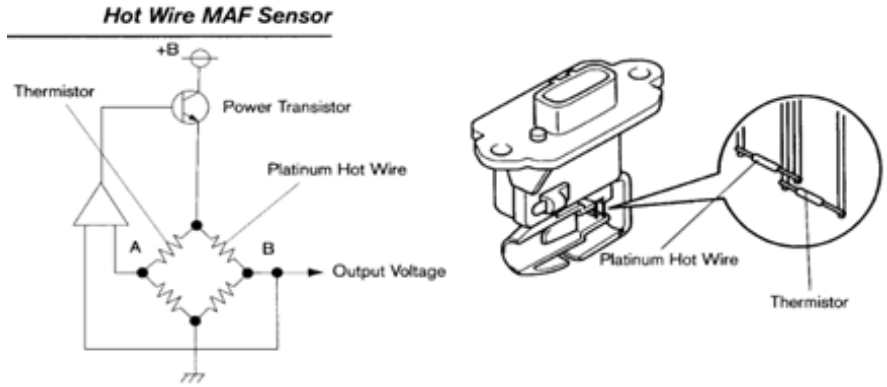


Figure 1. Hot Wire Mass AirFlow Sensor.

Another method that is commonly used for detecting airflow in HVAC systems is using a differential pressure sensor. Essentially, the sensor measures the velocity pressure of air at its location in the duct. This data can then be translated into a flow velocity using a modified version of the Bernoulli Equation. This will be shown in the Airflow Sensor portion in the Technical Design Section of this report.

As for the temperature and humidity sensors, there is a large range of digital products available on the market that will meet our needs. These can provide accurate data, have low power consumption, and readily interface with most microprocessors.

2. Requirements

A proper HVAC monitoring device must be able to meet the needs of consumers in order to be a worthwhile product. With this in mind, the marketing and engineering requirements for the monitoring system may include those listed below in **Table 1**.

Table 1. Engineering and Design Requirements.

Marketing Requirements	Design Requirements	Rationale
The device must be self-powered to incur no additional cost to run the HVAC system.	The device must not impede airflow or reduce the overall effectiveness of the system in any appreciable way.	This combination of requirements guarantees that the efficiency of the HVAC unit is not affected by the device.
	The device must harvest and store enough energy to supply its own energy needs and operate properly.	
The device must not require constant or continuous maintenance.	The device must provide data about its current functional status, including charge rate and battery life.	By constantly checking for and finding itself "as fully functional," the user will know that the device requires no maintenance.
The device must be durable enough to withstand the sometimes harsh conditions of an HVAC system.	Each component within the system must be able to operate between 0°C and 50°C.	HVAC units operate at a wide range of temperatures and the device must remain functional at all times.
The measured data must be accurate and reliable, even within a wide range of HVAC operating conditions.	The device must sample data at a reasonable interval (1 sample/5 minutes)	The data must be useful, and so it must be collected at regular intervals with a standard of accuracy.
	The measured temperature must be within 0.5°C of actual.	
	The measured air flow must be within 20 CFM of actual.	
	The measured humidity must be within 5% of actual.	
The system must be able to operate on 1.5 ton to 5 ton HVAC systems.	The device must be small enough that it fits in standard HVAC units without significantly affecting airflow. (Current testing unit is 18"x24", which is standard length and width of most major rectangular air ducts.)	The range 1.5 to 5 tons was selected because it covers 95% of active HVAC systems [16].
The device must be able to function as a retro-fit for existing HVAC units.	The system must be easily mountable.	The device's marketability depends on its ease of use in existing HVAC systems.

2.1 Engineering-Marketing Tradeoff Matrix [TM]

Below is a matrix illustrating the engineering and marketing tradeoffs, along with their inter-correlation.

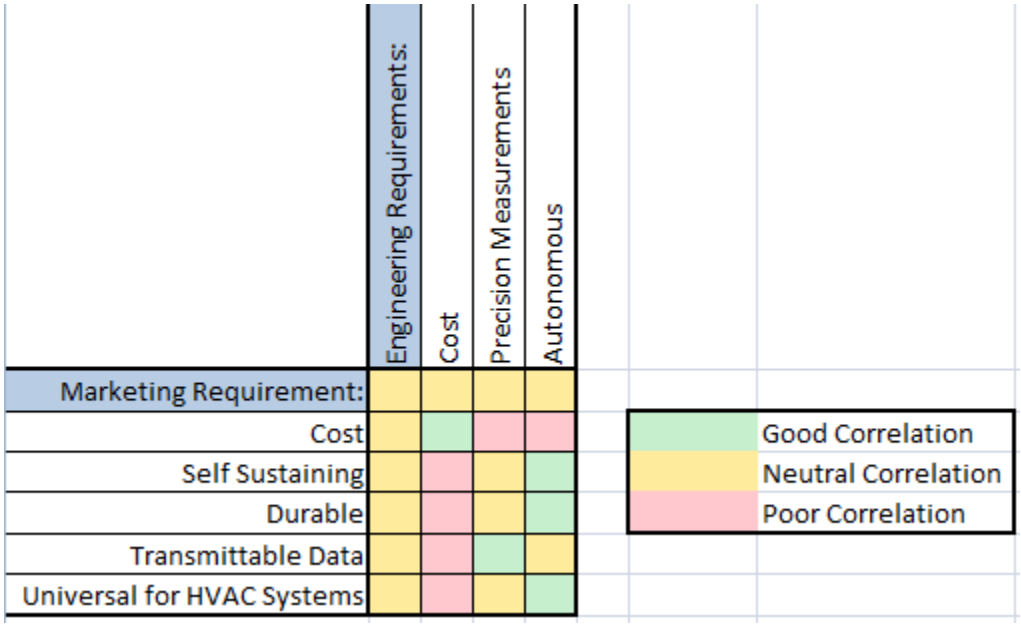


Figure 2. Engineering-Marketing Tradeoff Matrix.

2.2 Objective Tree [TM]

The objective tree, shown below in **Figure 3**, presents a break-down of each marketing requirement into subcategories, each with a weight associated with their level of importance.

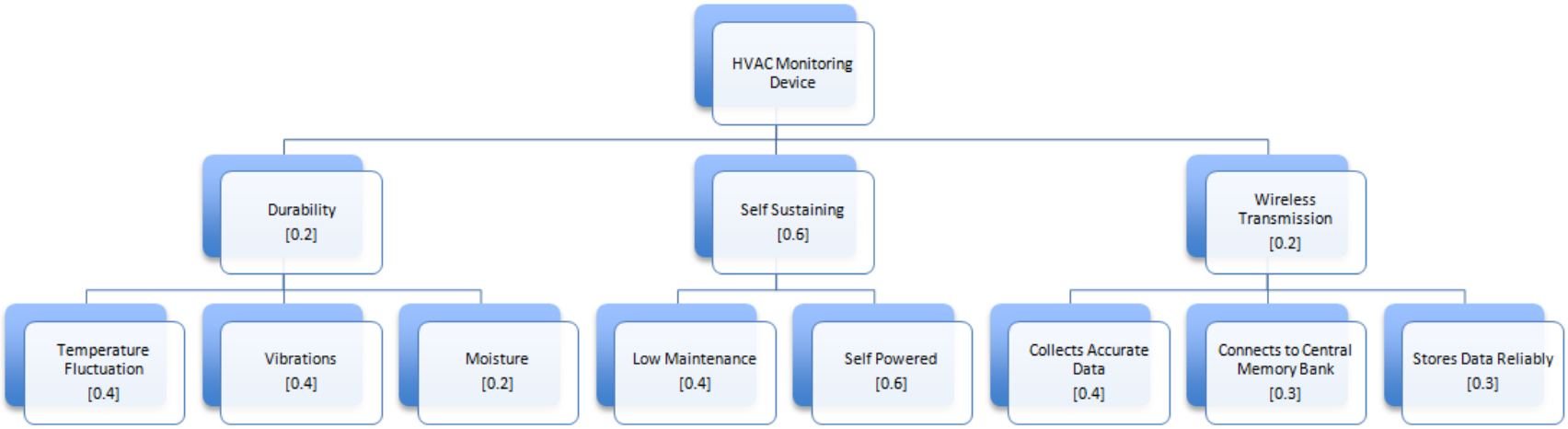


Figure 3. Objective Tree.

3. Technical Design

3.1 Power Overview [TM]

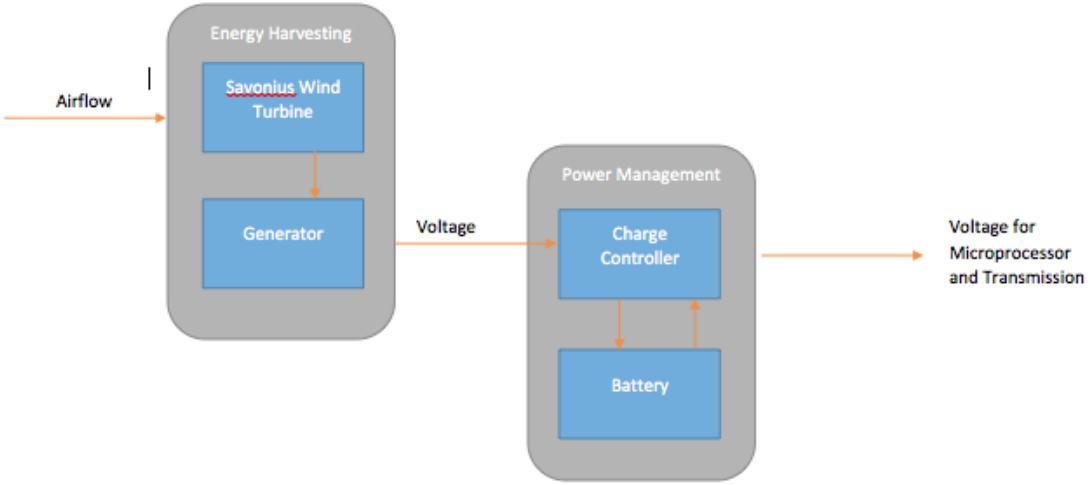


Figure 4. Level 2 Energy Harvesting Block Diagram.

Shown in **Figure 4** is the Level 2 Block Diagram for energy harvesting and power generation at the mouth of the air duct. The turbine will use the system airflow as an input to generate the torque needed to drive the DC generator to produce a DC output voltage. This voltage will then be used by the charge controller to manage the battery charging. The charge controller will also communicate the state of charge and the charge rate of the battery to the microprocessor.

At the beginning of the design process, a rigid fan motor was considered. While the fan motor was able to generate a voltage, it was greatly reducing the airflow from the HVAC unit into the duct work. From there, the most effective form of energy harvesting in the air duct was determined to be achieved by placing a Savonius wind turbine inside the opening directly above the HVAC unit. This positioning and design will allow the turbine to generate the greatest amount of voltage for operations and transmissions with the smallest amount of disturbance to the HVAC system. This design is also self-sufficient, meeting the requirement of not needing constant maintenance. The reasoning behind this decision and the proposed design of this feature will be explained in much further detail later in the Technical Design Section.

The charge controller is a necessary component in the subsystem for the safety and continuous charging of the battery, regardless of the amount of generated DC voltage. Because of the airflow level, the voltage generated would only be a percentage of the capacity of the battery. In order to increase the amount of voltage generated from airflow, and thus the turbine, a charge controller will be used. The charge controller will also manage the charge/discharge cycles of the battery in order to keep the battery from overcharging or dipping below a safe charge threshold. Overcharging or allowing a battery to go below a safe charge threshold would destroy the battery. There are built-in internal thresholds in the battery that, when breached, will allow the charge controller to stop the battery from operating. With the battery below operating threshold, the charge controller is in a state of charge, where the impedance is low, allowing for the generator to influence the charging of the battery. When the battery is at a fully charged state, the charge controller status see's a high impedance, which allows the charge controller to stop charging the battery while in high impedance state.

While the considerations for the battery were not as complex as those for other subsystems, they were some of the most important in that a rechargeable battery will be absolutely necessary to fulfill the requirement of self-sufficiency. Further, given the relatively low data transmission rate of the system, the charge capacity of the battery will be such that a small-sized battery would accommodate the power and physical needs of the design. That is, a smaller scale battery would adequately meet the power requirements of the proposed system without its physical size impeding airflow.

3.2 Power Generation [TG]

Selecting the proper turbine for the application is crucial to the project's success. There are many different types of rotors which can be used as wind turbines, on many different economies of scale. Each type of rotor comes with its own advantages and drawbacks pertaining to weight, airflow interference, operating efficiency, torque generated, and susceptibility to overheating.

Wind power operates under the principle that air movement carries a finite amount of energy, dependent upon the density of air at the point it pushes against the rotor, the cross sectional area of the rotor, and the speed of the air as it makes contact with the rotor. The general equation for wind power is given by

$$P_r = \frac{1}{2}\rho A U^3 C_p \eta_{mech,}$$

where P_r is rotor output power, ρ is air density, A is rotor swept area, C_p is a non-dimensional power coefficient representing power extracted by the rotor, and η_{mech} is the mechanical efficiency of the rotor and structure, [13]. The efficiency of a wind turbine is a function of mass flow rate through the apparatus and the pressure differential between air at the input and the output.

3.2.1 Betz Limit [TG]

Due to the law of conservation of energy, any kinetic energy from the airflow which is converted to mechanical power cannot return to the system as airflow, but is instead converted to electricity or lost as heat. If all of the kinetic energy from air movement through the turbine were extracted as mechanical energy, the kinetic energy at the output of the system would be 0, thus preventing old air from moving out of the turbine and allowing new air to move in. In order to keep air moving through the turbine, the air speed must be greater than zero after it moves past the rotors. This limit on turbine efficiency was published by German physicist Albert Betz in 1919, and is known as Betz's Law, [12].

Betz's Law shows that as air flows through a given area, if it is to keep moving in the same direction after being slowed by a turbine it must spread out to a wider area. As a result, geometry limits any turbine efficiency to 59.3%. The non-dimensional power coefficient of the equation above therefore has a maximum value of 0.593, and is determined based on the geometry of the turbine in use. **Figure 5** shows the power coefficients of various turbines, which have been determined based on the geometry of these devices. In this graph, 'λ' is the tip speed ratio. This is the relationship between the wingspan of the motor to its speed of rotation. C_p is the power coefficient of the turbine, as a function of this ratio.

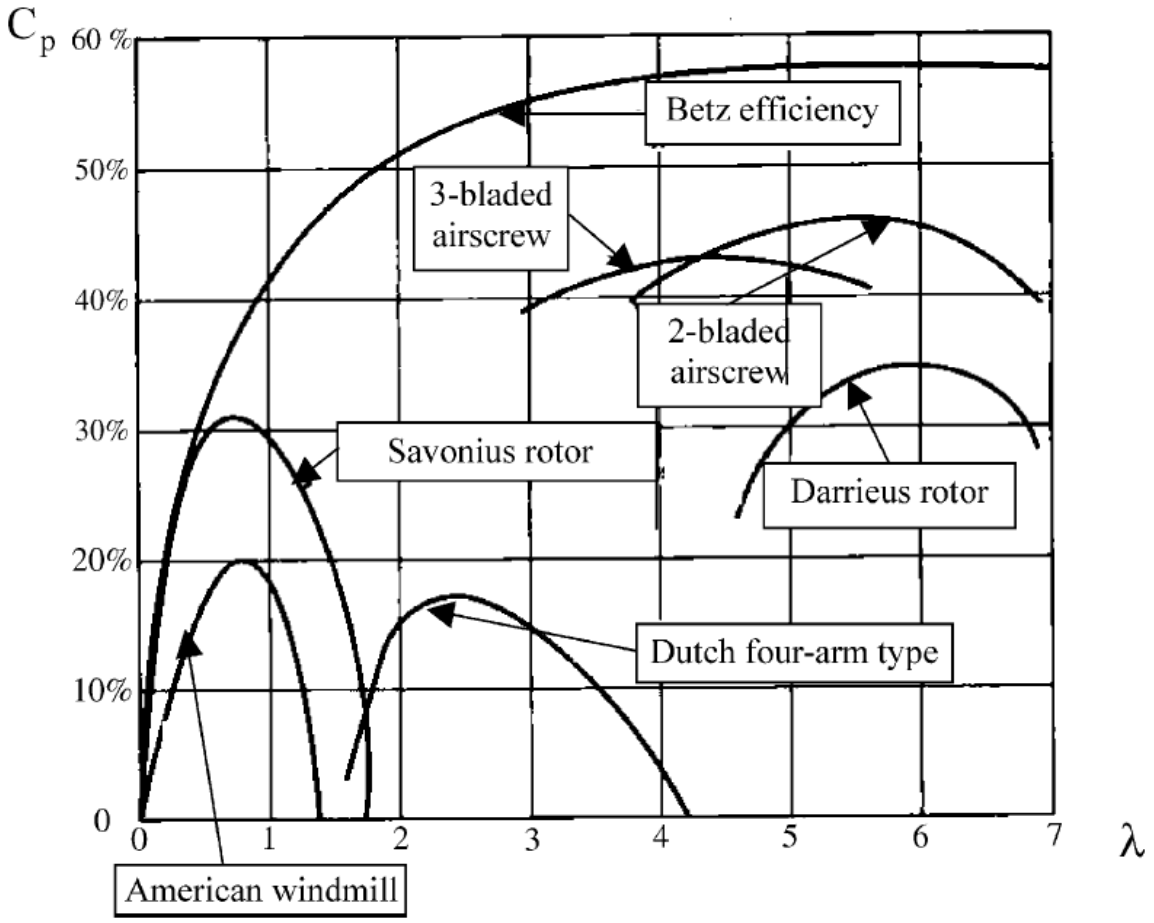


Figure 5. Power Coefficients for Different Types of Wind Turbines, [12].

As the value of C_p increases, the airflow interference of the system increases. In most wind turbine applications, the objective of the system is to maximize C_p, and get as close to the Betz limit as possible. However, the objective of this project is to disrupt airflow as little as possible, so the heating and cooling capabilities of the unit remain unaffected. For this reason, the Betz limit does not necessarily need to be approached, as long as enough power is generated for the device.

3.2.2 Turbulence [TG]

A turbine with better C_p performance would require less cross sectional area to produce the same amount of power. However, a greater $C_p$value comes at the cost of an increased level of turbulence behind the rotor. **Figure 6** and **Figure 7** show screenshots of CFD simulations of a Savonius Rotor and a standard 3-blade airscrew, at the point which they appear to be generating the largest amount of air velocity deficit.

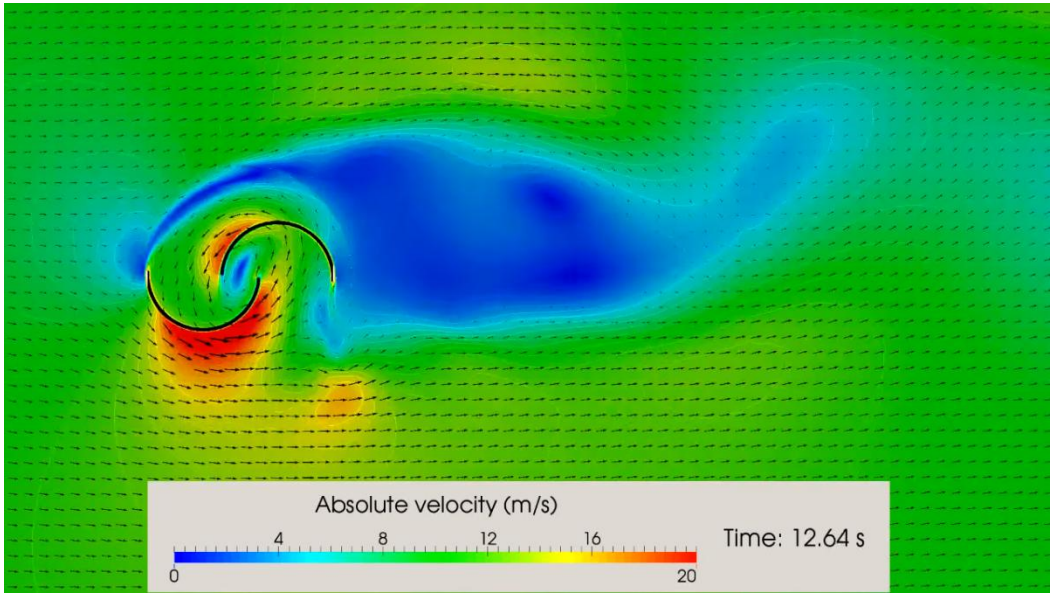


Figure 6. Turbulence Profile of the Savonius Rotor.

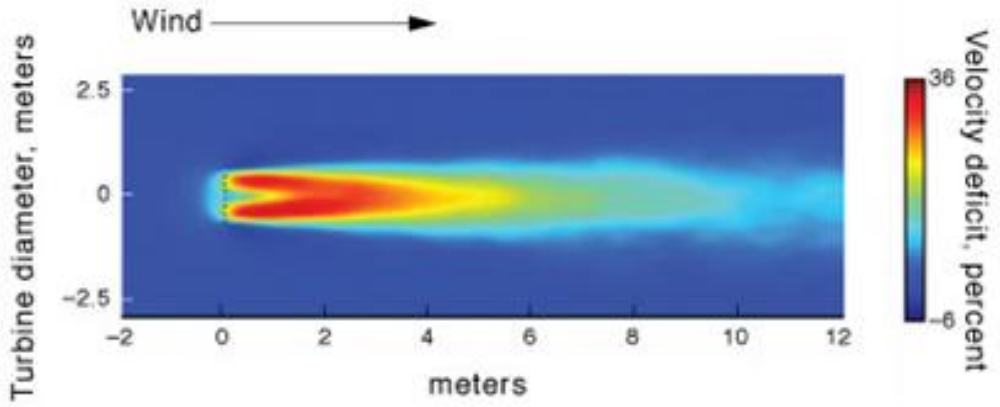


Figure 7. Turbulence Profile of Standard Windmill.

The turbulent flow produced by the Savonius Rotor (a vertical wind turbine design) is concentrated in an area approximately 3.5 times the size of the cross sectional area of the turbine itself, [14]. However, the turbulent area behind the standard windmill design covers well over 10 times the cross sectional area, and does so at an approximately equivalent intensity as that of the Savonius design [15].

It should be noted that the Savonius rotor simulation was modeled on a much smaller scale than the standard windmill design was, and that both of them are larger than any sort of turbine that would be used for this project. However, because the geometry of these devices is what determines the $C_p$values as well as the turbulence they produce, these results are scalable and still apply to smaller versions of these turbines. A horizontal turbine design is more appropriate to use as a generator in a duct, given the constraints of the project.

3.2.3 Other Considerations [TG]

In addition to concerns about airflow, a component to motor selection was its propensity to be overspun by the amount of airflow moving through it. Sending too much current through the coils of the motor could cause them to overheat, and the system could catch fire. Most wind generation devices have a built-in controller to manage overspin; current may be used to induce an emf in the motor to resist motion, for example [15]. However, the HVAC monitoring device is intended to be entirely self-sufficient. As such, it is important that it is able to operate at a wide range of air speed conditions with little to no maintenance.

Therein lies the largest advantage of the Savonius turbine, over the other types of small turbine devices; it is geometrically impossible for the device to overspin [11]. Because of its relatively low C_p rating, even across a very large range of wind speeds, the device geometry prevents it from gaining enough momentum to require a separate controller to curb it. Because the operational conditions of the proposed design vary from 0-1400CFM, with the exception of the Savonius wind turbine, any other turbine would need an additional controller to ensure that it does not break apart or overspin.

3.2.4 Decision Matrix [TG]

The turbine design was ultimately decided upon using a decision matrix, which visually represents the pros and cons of each type of turbine. From the decision matrix shown in **Figure 8** below; with all factors weighed in, it can be clearly seen that the Savonius turbine is the best candidate for the proposed HVAC Monitoring Device.

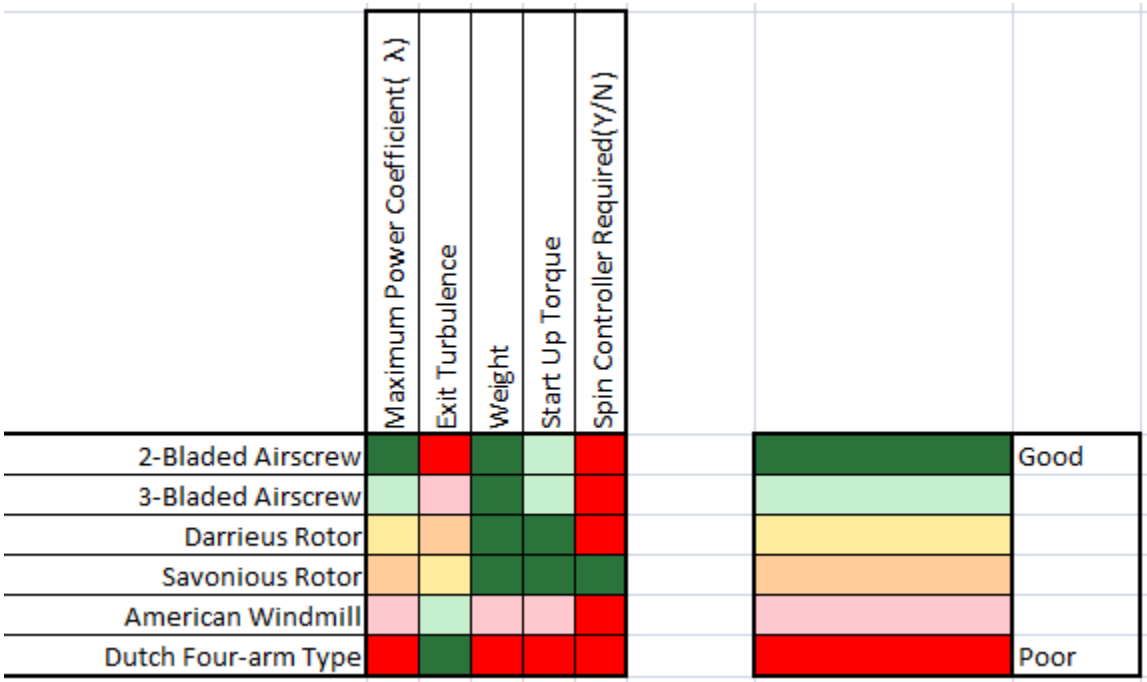


Figure 8. Turbine Decision Matrix.

3.2.5 Savonius Turbine Design [TG]

As illustrated in **Figure 8** above, using a Savonius turbine is the best choice for the proposed project. However, there are still many factors which will affect the performance of the turbine, aside from the type of rotor which is attached to it. The orientation of the turbine in the duct, the material used to make the turbine, as well as the construction of the turbine itself, are just some of the factors which must be considered. This section explores the physical considerations associated with the design of the turbine, which will drive the choice and design of the energy harvesting device.

3.2.6 Orientation of the Turbine in the Duct [TG]

In order to meet the marketability requirements of the project, the airflow moving through the duct cannot be blocked in any way that it is detrimental to the heating or cooling capabilities of the HVAC system. The average residential HVAC system blows 400CFM/ton, and can be sized anywhere from 1.5 to 5 tons [16]. These systems are considered fully functional within a range of 350CFM/ton-500CFM/ton, meaning that the rotor can afford to take a maximum of 12.5%, or 50CFM, of the airflow from a 400CFM system and the consumer will notice no difference in the climate control capabilities in their home.

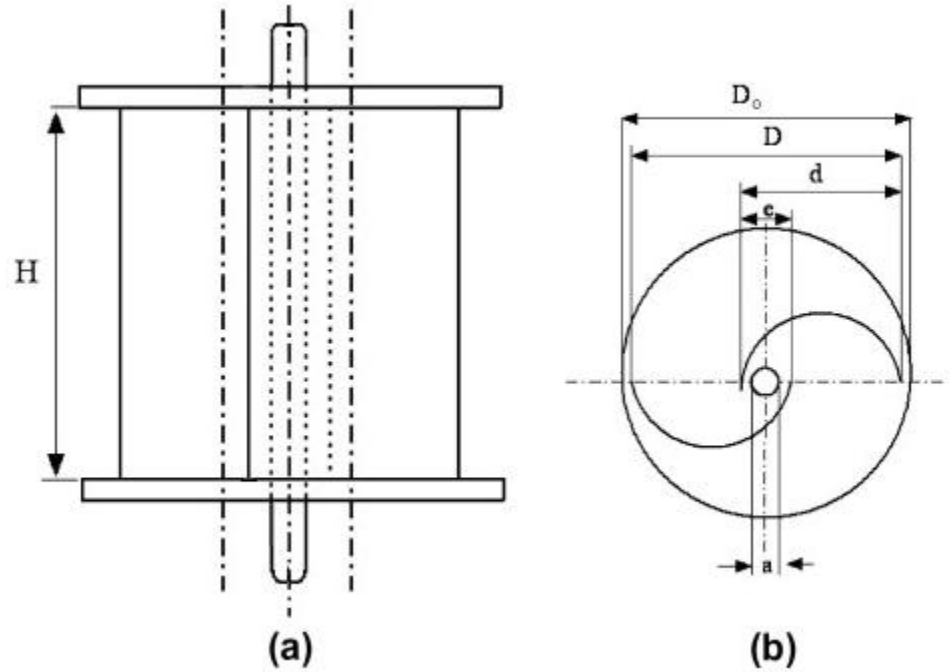
A Savonius turbine may transform a maximum of 30% of the airflow it disrupts into electrical energy, as shown in **Figure 5**. This means that it could have a cross sectional area which covers up to 20% of the duct space and the system would still be able to provide adequate heating and cooling capabilities from the unit. This would only use ~6.6% (found by multiplying 30% of 20%) of the allowable 12.5% of airflow and leave plenty of headroom for error.

The standard size for an HVAC unit duct connection is 18” x 24”, which is approximately 2780cm^2. The Savonius rotor would then have a maximum cross sectional area of about 556 cm^2. However, the turbine also includes a DC generator which has a cross sectional area of 26.24 cm^2. Accounting for the airflow blocked by the generator, the maximum area the rotors could cover then becomes about 530 cm^2.

Running the motor along the edge of the duct was determined to be the best place for the rotor, as it provides the greatest opportunity for free-flowing air to disrupt the turbulence caused by the turbine. Placing the motor in the center of the duct would generate two small areas of free-flowing air, each tucked between an area of turbulence and the sides of the duct, which have friction forces that slow the speed of fluids that move through them. By placing the device nearer to one edge of the duct, there is a larger airstream whose flow remains uninterrupted.

3.2.7 Turbine Design [TG]

There are three primary constraints to the design of the turbine, in addition to the power requirements; namely, the dimensions of the duct, the cross sectional area which the turbine is allowed to cover, and the turbine location – the turbine must be near one of the edges of the duct to permit as much free-flowing air as possible. **Figure 9** shows the dimensions considered for a Savonius rotor design.



Parameters:
d – Diameter of rotor wing [m]
D – Wing spread of rotor [m]
e – Rotor spacing [m]
H – Height of turbine [m]
D_o – Diameter of end plates [m]

Figure 9. Savonius Rotor Dimensions, [11].

The turbine will be tested at the University of Akron Electrical and Computer Engineering test facility, which is approximately 1000ft above sea level; this means that the air density will be ρ=1.13731066 kg/m^3. The sweep area of the rotor blade is A = 0.53 m^2, and C_p = 0.3 because it is the maximum possible power coefficient for the geometry of a Savonius rotor.

The overlap *e* of the two rotor wings is an important aspect of the design. It has been shown by O. Mojola [17] and J.L. Menet [18] that the optimal overlap for a Savonius wind turbine is about 25% of the rotor wing diameter *d* to generate the maximum amount of power from a given air flow rate. The amount of power the rotor receives from the air is driven by the equation

$$P_r = \frac{1}{2}\rho A U^3 C_p \eta_{mech}.$$

For our purposes, we assume that the mechanical efficiency of the device is high (~1), and that the C_p value is at its maximum, 0.3.

Rotational speed is defined as

$$n = \frac{60}{2\pi}\omega,$$

where ω is the angular velocity of the rotor, in radians per second. For a Savonius turbine,

$$\omega = \lambda * \frac{\mathrm{v}}{\mathrm{r}},$$

where λ is the tip-speed ratio, v is the velocity of air at the tip of the turbine, and the radius of the turbine $r = D/2$. The tip-speed ratio is a property of turbines that is determined geometrically and verified experimentally; for a Savonius turbine, λ=1, [12].

The torque at the rotor shaft is given as

$$\tau_s = \frac{P_s}{\omega}.$$

Table 2 and Table 3 were generated to compare the theoretical power output of various sized rotor wings. The cross sectional area was capped at a conservative 0.5 m^2, again to help maximize free flowing air in the duct. Because there is a range of HVAC unit sizes, calculations were done for the minimum size unit (1.5 tons, 300CFM/ton) in **Table 2**, and for a very large unit (5 tons, 500CFM/ton) in **Table 3**.

Table 2. Small HVAC Unit Power Calculations (v = 1m/s).

H [cm]	d [cm]	e [cm]	D [cm]	r [cm]	ω [rad/s]	n [rpm]	P_s[W]	τ_s[N·m]
10	26.286	6.5715	46	23	4.3478	41	0.0153	0.003519021
21	12	3	21	10.5	9.523	90	0.0153	0.001606637
25	10.286	2.571	18	9	11.1111	106	0.0153	0.001377001
35	7.428	1.857	13	6.5	15.385	146	0.0153	0.0009945
50	5.1429	1.286	9	4.5	22.22	212	0.0153	0.0006886

Table 3. Large HVAC Unit Power Calculations (v = 3m/s).

h [cm]	d [cm]	e [cm]	D [cm]	r [cm]	ω [rad/s]	n [rpm]	P_s [W]	τ_s[N·m]
10	26.286	6.5715	46	23	13.043	124	0.13818	0.010594188
21	12	3	21	10.5	28.571	272	0.13818	0.004836373
25	10.286	2.571	18	9	33.333	318	0.13818	0.004145441
35	7.428	1.857	13	6.5	46.154	440	0.13818	0.002994
50	5.1429	1.286	9	4.5	66.667	636	0.13818	0.002072

Using this theoretical analysis, it was determined that a 35cm x 13cm Savonius Rotor will be sufficient to spin the DC generator at a rate which will power all of the on-board devices. **Figure 10** is a 2D representation of what the generator and turbine will look like in the HVAC opening.

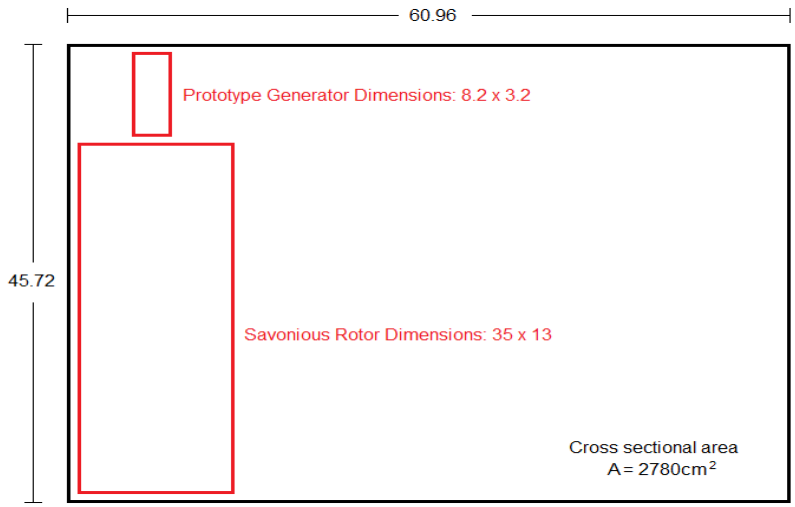


Figure 10. Final Energy Harvesting Dimensions.

The intent is to use the University's 3D printing services to 3D-print the turbine, because it is relatively inexpensive, and will allow for obtaining the precise measurements required for this application. While plastic is a lightweight material, which will spin easily in the airstream, it is also durable enough to handle any small debris which may be blown through the unit.

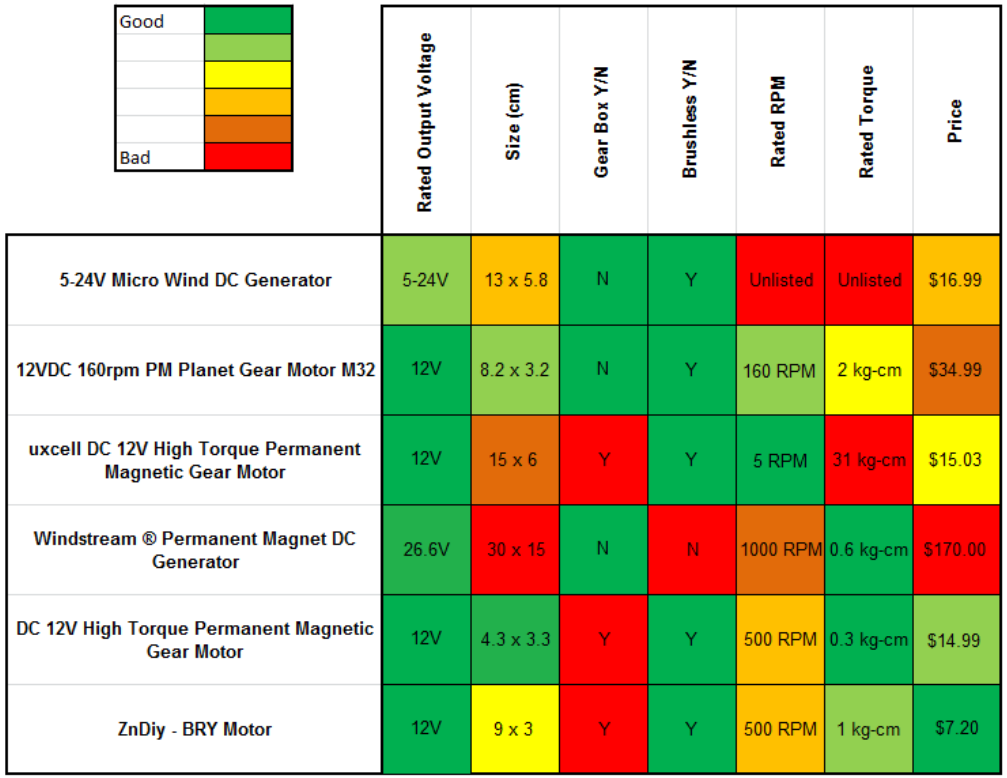
3.2.8 DC Motor Selection [TG]

Part of the wind turbine design includes the use of a DC motor to generate electricity. By using the mechanical energy of the turbine to push the rotor of a DC motor, current may be induced in its coils and therefore used to power our device. The primary factors which had to be considered in deciding on a motor were the rated output voltage, rated RPM, and rated torque.

Spinning a motor at its rated RPM can produce approximately its rated voltage [12]. For wind generation it then follows that it is best to use a motor with a high rated voltage, and low RPM. However, this comes at the cost of increased torque to start and keep the motor running. The inclusion of a gear box in the construction of the motor also provides mechanical resistance which, in turn, increases the torque required to get the motor going.

It has been shown that permanent magnet DC Motors are the ideal choice for use as a generator in wind power applications [12]. These motors can either include brushes or be brushless. However, a brushless motor would never need brush replacement, and since the purpose of this project is to minimize maintenance requirements, it was determined that this would be the best option.

To help balance these considerations, a decision matrix was used to help decide on a motor for the construction of the proposed device, shown in **Figure 11**.



	Rated Output Voltage	Size (cm)	Gear Box Y/N	Brushless Y/N	Rated RPM	Rated Torque	Price
5-24V Micro Wind DC Generator	5-24V	13 x 5.8	N	Y	Unlisted	Unlisted	$16.99
12VDC 160rpm PM Planet Gear Motor M32	12V	8.2 x 3.2	N	Y	160 RPM	2 kg-cm	$34.99
uxcell DC 12V High Torque Permanent Magnetic Gear Motor	12V	15 x 6	Y	Y	5 RPM	31 kg-cm	$15.03
Windstream ® Permanent Magnet DC Generator	26.6V	30 x 15	N	N	1000 RPM	0.6 kg-cm	$170.00
DC 12V High Torque Permanent Magnetic Gear Motor	12V	4.3 x 3.3	Y	Y	500 RPM	0.3 kg-cm	$14.99
ZnDiy - BRY Motor	12V	9 x 3	Y	Y	500 RPM	1 kg-cm	$7.20

Figure 11. DC Motor Decision Matrix.

3.3 Battery Selection [TM]

Power generation will remain constant as air flows in the HVAC unit. However, the demand for power will not be continuous. In order to not waste a large amount of the energy generated by the wind turbine, a battery will be used to store the energy between the wake-up and sleep cycles of the device. This provides a dual advantage; first, energy can be generated slowly, over the course of a few minutes in order to power the device. Second, it ensures that the power received by the devices is consistent, and at the correct voltage

and current levels. This will ensure that the components will not be subjected to electronic failure, and improve the longevity of the device.

Prior to deciding on a battery, multiple versions of batteries were considered. The ideal battery for this application would be one having the highest storage capacity while requiring a very small physical shape. The downside to having a high power requirement is that batteries tend to exponentially scale up in size as their voltage capacity increases. By calculating the power requirements of the proposed system, a voltage that could be used for targeting a suitable battery for the system was found.

The battery that was chosen is the Adafruit Lithium Ion Polymer Battery – 3.7v 150mAh. This battery will be able to handle the voltage necessary to perform all of the tasks that the proposed device will need, as well as be small enough to be contained inside the air duct without causing issues due its size. The Adafruit battery chosen has a range of 3.7-4.2V, depending on the current state of charge. With the generator connected to the power system, the battery will have a constant charge while in a state of low resistance from the charge controller. This allows the battery to be safely charged up to its maximum level while the energy harvesting device is continuously operating.

Table 4. Battery Level 2 Requirements.

Module	Battery – Adafruit Lithium Ion Polymer Battery – 3.7v 150mAh
Input	Voltage – from Charge Controller
Output	Voltage – to Battery
Functionality	Battery power will be used to first power up the internal processes of the system, like the microprocessor and sensors, and then to power up the communication devices during data transmission.

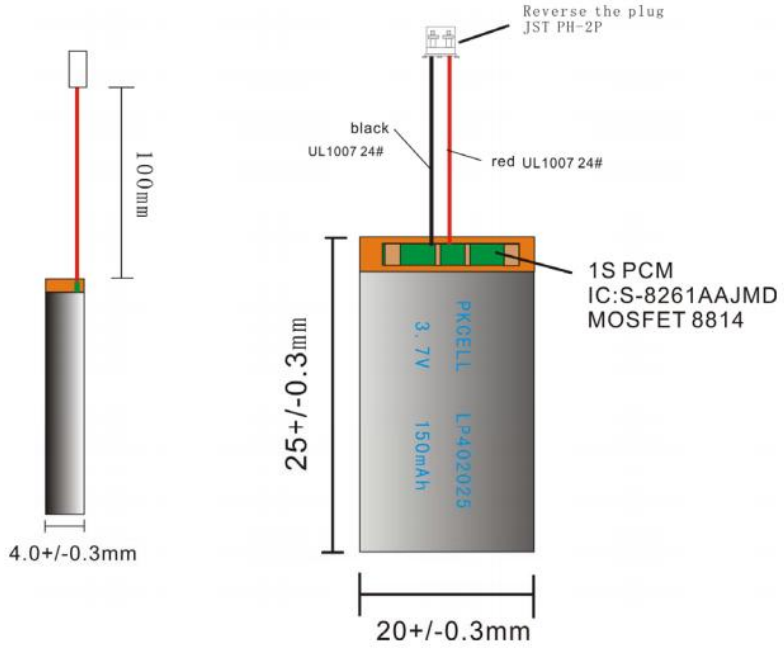


Figure 12. Adafruit Lithium Ion Polymer Battery – 3.7v 150mAh.

3.4 Charge Controller [TM]

The charge controller will be used to optimize charging of the battery as well as keep a continuous charge rather than a maximum charge on the battery. The need and desire for a charge controller are to enable the energy-harvesting device to consistently charge the battery instead of having the turbine capped at maximum generation. By introducing a charge controller, the energy-harvesting device has a structured form of limiting incoming power as well as ensuring correct charging and discharging procedures with the battery. For incoming power, the voltage will be regulated down to 4.25V. While the regulation is necessary to allow for a more stable charge, the trade-off to rectifying the voltage is that the system could potentially overheat. Taking that into account, the turbine and harvesting unit is placed in the duct in such a way that the possible voltage is close to the rectified 4.25V. This allows for the charge controller to still maximize its potential rectification while not overheating to a point of interference in data collection.

The charge controller is an individual component that comes directly associated with the battery. The Adafruit Pro Trinket Lithium Ion Polymer Backpack is a direct component feature that Adafruit provides with the battery as a package. Therefore, the interfacing between the charge controller and the battery requires no additional components.

Table 5. Charge Controller Level 2 Requirements.

Module	Adafruit Pro Trinket Lithium Ion Polymer Backpack
Input	Voltage – from Limiter
Output	Voltage – to Battery
Functionality	The charge controller will be used to maximize battery usage. Since the airflow will generate a voltage from the turbine, through the DC generator, the voltage supplied to the battery will need to be regulated.

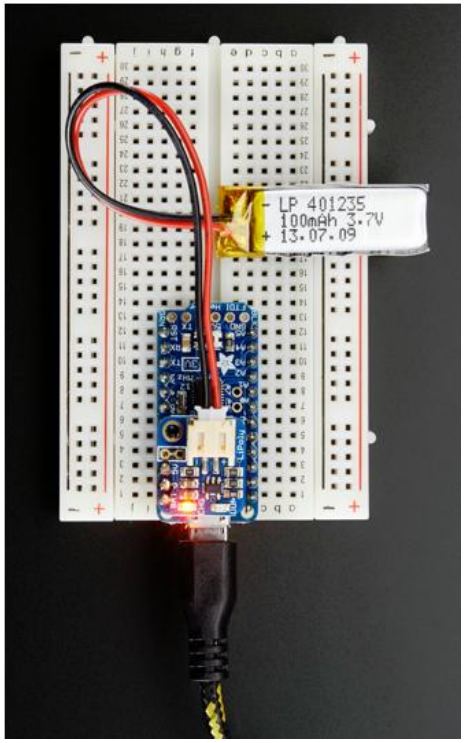


Figure 13. Adafruit Pro Trinket Lithium Ion Polymer Backpack with Zener Diode Model.

3.5 Temperature and Humidity Sensor [RG]

The decision for a temperature and humidity sensor was a very easy one to make because of the number of options that were available with digital output. The one that we have chosen for this project is the Adafruit DHT22. Adafruit is a reliable, competitively-priced store that provides a large amount of information about their products at no extra charge. For this sensor, it has a step-by-step process of how to set the sensor up for testing, libraries that are free to download to interface with devices such as Arduino, and sample code of how to use it. The sensor uses a capacitive humidity sensor and a thermistor to collect data within 2%-5% humidity and ±0.5°C, thereby fulfilling our accuracy requirements. The power requirement is only about 9 mA while requesting data, which should not be an issue with the power production expected. The only downside to the sensor is that it is only able to take measurements every 2 seconds, so data can be up to 2 seconds old when transmitting. We do not expect to transmit data at a rate anywhere near this rate, so it should not be an issue in this project. The sensor functionality table is shown in **Table 6** below, providing a more compact view of some of the main parts of the sensor.

Table 6. Temperature and Humidity Sensor Level 2 Requirements.

Module	Adafruit DHT22 temperature-humidity sensor + extras
Input	Voltage – from Battery, Temperature and Humidity Readings
Output	Data – to Microprocessor in digital form
Functionality	The temperature and humidity sensor will be used to gather current data in the HVAC duct and output it to the microprocessor.

3.6 Airflow Sensor [RG]

While mass airflow sensors seemed like it would be a good solution to measure airflow, there are a number of issues with the options that are currently available on the market. Honeywell's turned out to be the only airflow sensors that run on the 3.7V or 5V battery to be used for our power supply options. From their selection, power consumption and physical size seemed to be large issues. The smallest versions available from Honeywell were the Zephyr line of products. The operating conditions of this line of sensors can be found below in **Figure 14**.

Characteristic	Parameter
Supply voltage 3.3 Vdc 5.0 Vdc	 3.3 Vdc ±10% 5.0 Vdc ±10%
Power: 3.3 Vdc 5.0 Vdc	 40 mW max. 65 mW max.
Compensated temperature range	0 °C to 50 °C [32 °F to 122 °F]
Operating temperature range	-20 °C to 70 °C [-4 °F to 158 °F]
Accuracy	See Table 3.
Total Error Band (TEB)	See Table 3.
Null accuracy	0.1 %FSS
Response time	1 ms typ.
Resolution	12 bit min.
Start-up time	17 ms
Warm-up time	30 ms
Calibration media	gaseous nitrogen
Bus standards	I^2C fast mode (up to 400 kHz)
Null stability	Null will not deviate beyond the specified TEB.
Reverse polarity protection	no

Figure 14. Honeywell Zephyr SCCM Sensor Operating Conditions.

As can be seen from **Figure 14**, power consumption is 40 mW max at 3.3 V supply and 65 mW max at 5 V supply. Further research revealed that this is a low estimate and that the device would likely use even more power than this. With a 30 ms warm-up time before data can be acquired, this issue is only further exacerbated. In addition, the 750 Standard Cubic Centimeter per Minute, SCCM, max airflow available with this line of products is very low compared to the amount that an HVAC system would likely provide. Using the conversion factor that 1 CFM is equal to approximately 28317 SCCM (readily found online), this sensor could only measure up to 0.0625 CFM through it. Even with the small ports with which they measure the airflow, and scaling the size of the port to the size of the entire duct, the sensor would likely be constantly maxed out. Clearly, this sensor is unsuitable for even the smallest of residential HVAC systems, let alone the thousands of CFMs that larger industrial units can supply. It is much better suited to higher precision projects in small pipes that have slow-moving gases.

Looking into the next size of the Zephyr line by Honeywell, the scale problem is lessened, but the power consumption worsens. The operating conditions can be seen below in **Figure 15**.

Characteristic	Parameter
Supply voltage	3 Vdc to 10 Vdc
Supply current	20 mA max.
Power: 3 Vdc 10 Vdc	 60 mW max. 200 mW max.
Calibrated temperature range[1]	0 °C to 50 °C [32 °F to 122 °F]
Operating temperature range	-20 °C to 70 °C [-4 °F to 158 °F]
Full scale (FS) flow[2]	10 SLPM, 15 SLPM, 20 SLPM, 50 SLPM, 100 SLPM, 200 SLPM, 300 SLPM
Calilbrated flow range	0 to 10 SLPM, 0 to 15 SLPM, 0 to 20 SLPM, 0 to 50 SLPM, 0 to 100 SLPM, 0 to 200 SLPM, 0 to 300 SLPM
Calibration gas[3]	clean, dry air
Accuracy[4] 0 SLPM, 15 SLPM, 20 SLPM, 50 SLPM, 100 SLPM, 200 SLPM: 0 %FS to 14.3 %FS 14.3 %FS to 100 %FS 300 SLPM only: 0 %FS to 14.3 %FS (0 SLPM to 43 SLPM) 14.6 %FS to 100 %FS (44 SLPM to 300 SLPM)	 0.5% FS 3.5% reading 0.5% FS 3.5% reading

Figure 15. Honeywell Zephyr SLPM Sensor Operating Conditions.

From **Figure 15**, the power consumption at 3 Vdc can be seen as 60 mW max. This is even farther from the amount that we are likely to be able to provide than that of the SCCM line. As for the scale, by using the conversion factor of 1 CFM to 28.3 Standard Litre per Minute, SLPM, readily found online, the highest rating of 300 SLPM is still only able to measure about 10.6 CFM. Even after scaling the small port that it has attached to the amount of airflow the entire duct would be receiving, this would not be useable with industrial HVAC units. At this point, it seemed that a mass airflow sensor would be unlikely to work for our application, so other options need to be explored.

As alluded to earlier in the background of the report, the next type of sensor that was evaluated was the differential pressure sensor. This type of sensor measures the velocity pressure of air at its location in the duct, which is the kinetic pressure per unit volume of a fluid particle. This measurement can then be translated into a flow velocity. The formula that shows how to calculate the flow velocity from the velocity pressure can be seen below in **Figure 16** [19].

$$V = C * \sqrt{\frac{(2 * p_w * g_c)}{\rho}}$$

where:

V = Flow velocity (FPM)

p_w = velocity pressure (in. H_20)

ρ = density of air (lb_m/ft^3) (see table for estimates)

g_c = gravitational constant = 32.174 $lb_m*ft/lb_f s^2$

C = unit conversion factor (to feet and from in. H_20) = 136.8

Figure 16. Velocity Pressure to Flow Velocity Formula.

Multiplying the airflow velocity found from this formula by the cross-sectional area of the duct yields the flow rate at the location of the sensor. While not directly measuring the amount of air flowing in the duct, it should provide accurate enough data for monitoring purposes. In addition, differential pressure sensors require relatively low power, around 15 mW.

Due to the very particular set of parameters associated with our project, finding a sensor that would work for our exact purpose was a challenge. The first step in the process was to figure out approximately how much of a pressure difference a sensor in a typical HVAC duct would experience. In order to do this, the velocity pressure, p_w, in the system was found from the formula in **Figure 16**; that is,

$$p_w = \frac{\rho(\frac{CFM}{AC})^2}{2g_c},$$

where:

ρ = .0719 lb_m/ft^3 at 1000 ft above sea level
CFM = 1200 ft^3/min
A = 3 ft^2
C = 136.8
g_c = 32.174 $lb_m*ft/lb_f s^2$

Accounting for the testing conditions, and using the ratings for the actual HVAC unit that will be used in the project, yields a velocity pressure of 0.00956 in. H_2O. This is extremely low for most sensors, so the choices were fairly limited. In the end, the differential pressure sensor that was chosen is the SDP610-025PA by Sensirion. This particular sensor is commonly used for HVAC applications for a number of reasons. It has a range of ±0.1 in. H_2O, which is the closest that could be found to the range calculated. The sensor is able to function with 3.3 V and has a current draw of less than 6 mA, leading to a power consumption of around 20 mW. This should be acceptable for the amount of power we are expecting to harvest. As for accuracy, the reading is within 0.1Pa, which is equivalent to 0.004 in. H_2O. This amount should be good enough for calculating the airflow within acceptable limits. As for the interface, it uses I^2C, which should pair well with the microprocessor. The main downside to this sensor is that it requires approximately 50 ms warm-up time before "good" data is acquired. Even so, the power requirements should still be well within the acceptable range that has been determined. **Table 7** shows a functionality table for the airflow sensor in order to summarize some of the sensor more pertinent information.

Table 7. Airflow Sensor Level 2 Requirements.

Module	Sensirion SDP610-025PA
Input	Voltage – from Battery, Differential Pressure in duct (measured at two points)
Output	Data (Difference in airflow) – to Microprocessor in digital form as velocity pressure.
Functionality	The differential pressure sensor will be used to gather velocity pressure data in the HVAC duct and output it to the microprocessor. This can then be calculated into airflow in CFM.

3.7 Data Management [NO]

3.7.1 General Overview [NO]

The temperature, humidity, and differential pressure sensors will send the collected data to the microprocessor. The data will be first organized for easy transmission and then transmitted using a ZigBee device. A data collection computer will receive the data, send a confirmation back to the microprocessor, store the data into a local database, and display it graphically for the benefit of the user.

As stated before, a major concern for this project is power management, therefore in looking for a microprocessor and transmission devices, the emphasis was on low power solutions. Usually low power microcontrollers tend to provide inferior

processing power. However, for this project, not a significant amount of processing power is needed to carry out the tasks allocated to the microprocessor.

3.7.2 Microprocessor [NO]

We wanted a microprocessor that required a small amount of power but still had the processing power to adequately control our system. Early on, we discovered the Atmel ATtiny85. This microprocessor was found on an Adafruit product called the Trinket. The Trinket microprocessor board was a small and cheap development board that initially met our needs. The ATtiny85 has 8K of flash, 5 I/O pins, and with some simple modifications to the Arduino IDE, was Arduino-compatible. The ATtiny85, in low power mode consumed 0.52 mW, and in power-down mode consumed 0.18 µW at 0.1 µA. This power down mode saves the register contents and disables all chip functions until the next interrupt.

As the project design progressed, we discovered that we needed more I/O pins than the Trinket provided. We did not have to look far for a replacement. We decided on Adafruit's Pro Trinket microprocessor board. This development board is only a fraction bigger than the standard Trinket, contains 18 I/O pins, and houses the Atmel Atmega328P chip. The Atmega328P is a very popular chip on the Arduino platform; therefore the Arduino IDE will work with this board. In standby mode, the crystal Oscillator runs while the rest of the device is sleeping. This allows for a fast start-up and low power consumption. **Figure 17** shows the pinouts available on the Pro Trinket, while **Figure 18** shows the pins we are planning on using for our system. We are using the TX and RX pins for our transmission Device, an I^2C bus for our pressure sensor, and a few digital pins for our temperature/humidity sensor and charge controller.

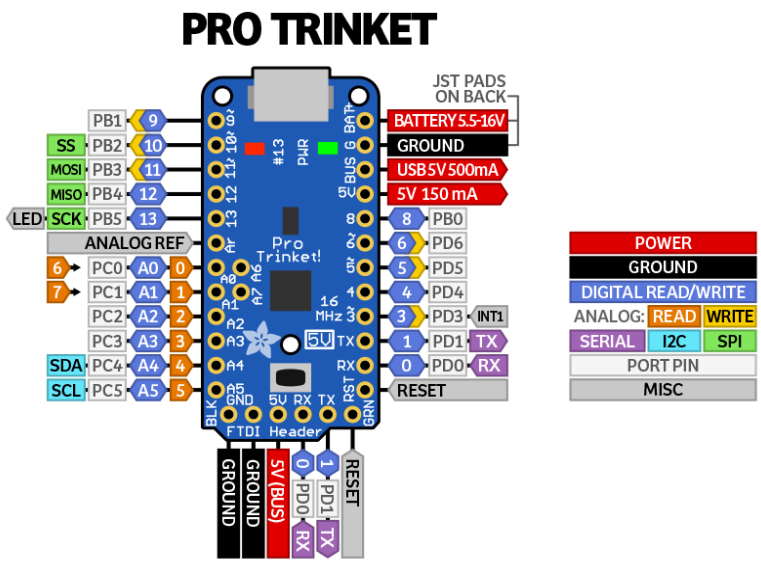


Figure 17. Pro Trinket Pin Diagram.

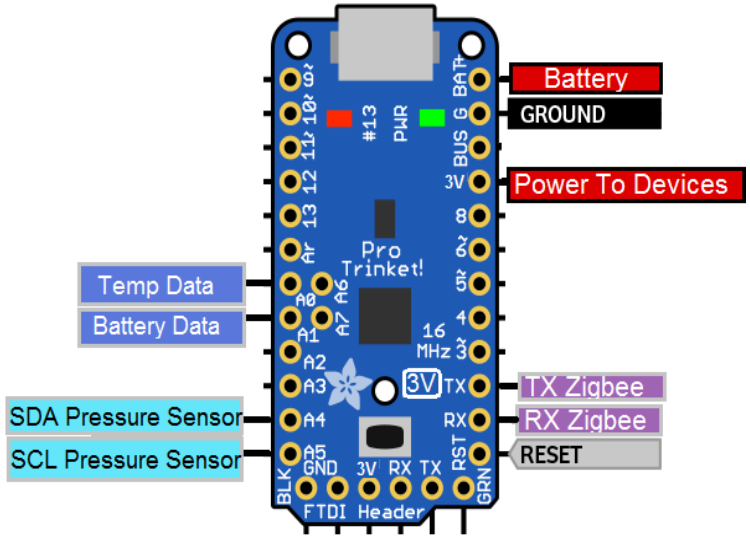


Figure 18. Pro Trinket Used Pins.

3.7.3 Transmission/Receiver [NO]

Originally, two transmission device options were considered; namely, ZigBee and MicroChip's MRF24J40MC. While both operate at 3.3V, there are a few key differences between the two devices. MicroChip's MRF24J40MC is a wireless transmitter which operates using more power, but over a longer distance. Dr. Ida expressed preference for the MRF24J40MC over the ZigBee alternative, and so we gave heavy consideration to this option. Below is a chart listing the relevant comparisons between the ZigBee and MicroChip devices. The sections highlighted in blue represent which transmitter is superior in each respect. The HVAC monitoring system will not require long distance communications, and keeping power demand low during transmission carries a lot of weight with this project. For these reasons, the ZigBee device appears to be a better fit for our application.

	ZigBee	MicroChip MRF24J40MC
Rx Power	132 mW	82.5 mW
Tx Power	132 mW	396 mW
Sleep Mode	13.2 µW	39.6 µW
Range	120 meters	1200 meters

Figure 19. ZigBee vs. MicroChip Device Power Comparison.

Upon deciding on ZigBee, and exploring their product line a "best match" to our project was found. ZigBee's two most common products are the XBee Series 2, and the XBee-Pro Series 2. And even though the Pro version has a greater indoor range, the standard XBee Series 2 consumes much less power. This made the choice to use the XBee Series 2 for our system quite clear. **Figure 20** highlights the key areas that led to our selection.

	XBee ZigBee Series 2	XBee-PRO ZigBee Series 2
Indoor Range	40 m	90 m
TX Power	116 mW	677 mW
RX Power	126 mW	155 mW
Sleep Power	3 uA	12 **u**A

Figure 20. Comparison Between ZigBee Devices.

3.7.4 Data Collection Center [NO]

A small computer or laptop will be the final destination of the data collected by the HVAC monitoring system. This computer will be connected to a ZigBee device and will receive data from the microprocessor. The data collection center will store the data in a local database. This computer will also display the collected data using a graphical user interface (GUI) with multiple graphs: Airflow vs. Time, Temperature vs. Time, Humidity vs. Time, Energy Stored in the battery, and it will log information about the rate of energy consumed vs. the rate of the energy generated. Below, in **Figure 21**, is a mockup of the initial design for the GUI. The final version of the GUI can be seen in **Section 5.6**.

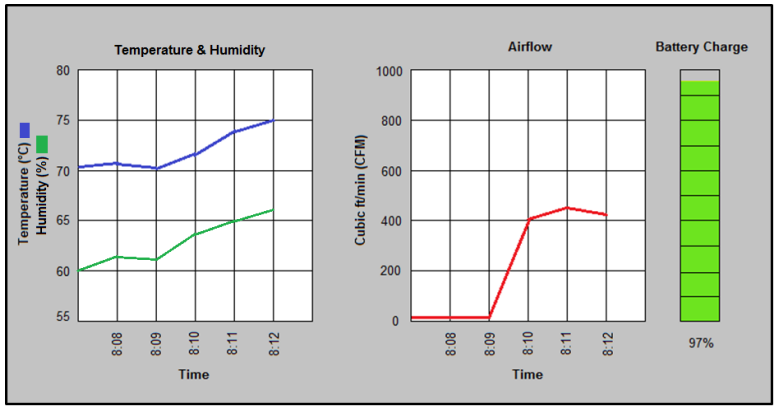


Figure 21. Mock Graphical User Interface.

3.7.5 Software [NO]

The system will contain two blocks of software; one on the microprocessor, and the other on the data collection center computer. The software on the microprocessor will be written in C using the Arduino IDE. The software on the computer will be written in C# using Visual Studio.

The microprocessor will wake from a sleep mode and send power to the sensors and the ZigBee. It will then wait and listen for sensor data. When data is received, the data will be prepared for transmission. Once the data has been organized, the microprocessor will transmit the data using the ZigBee. Once a confirmation is received from the data collection center, the microprocessor will de-energize the sensors and the ZigBee, and wait for the charge controller to signal that the battery is fully charged. The microprocessor will then "sleep" and wake from an internal timer set to our data sample time.

The data collection center will be idle until the unit receives a transmission from the microprocessor. Once the transmission is received, the data will be checked for completion using a parity bit. If the data is complete, it will send a confirmation back to the microprocessor, if not, the data center will request a retransmission. Once the data center receives complete data, the computer will organize the data, store the data, and display the data to the user. The data center will then wait for another transmission from the microprocessor. **Figure 22** shows the software flow through both the microprocessor and the data collection center (labeled PC). As mentioned before, there will be two software blocks; that of the microprocessor, and that of the data collection center. **Table 8** and **Table 9** describe the functionality of each block respectively.

Table 8. Functional Requirements for Microprocessor (Software).

Module	Microprocessor (Software)
Inputs	Sensor data - Temperature, Humidity, Air Speed Battery Data - Charge Level Transmission - ZigBee Confirmation
Outputs	Transmission - Organized Data
Functionality	Receive data from sensors and charge controller, organize it, and transmit it to the data collection center.

Table 9. Functional Requirements for Data Collection Center (Software).

Module	Data Collection Center (Software)
Inputs	Transmission - Data from Microprocessor
Outputs	User Interface - Data Graphs, Battery Level Database - Store Data
Functionality	Receives data from microprocessor, stores and displays that data for the user.

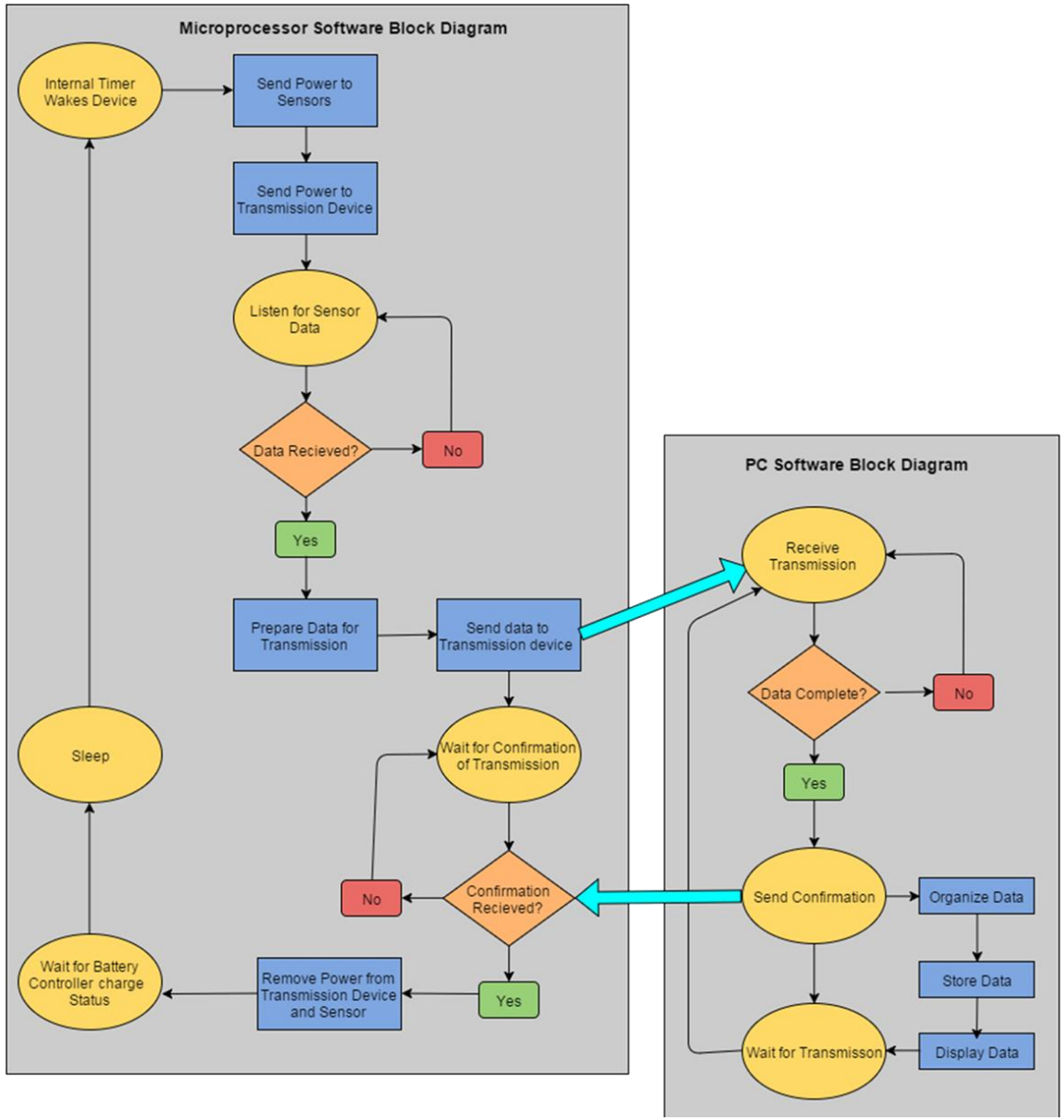


Figure 22. Software Block Diagram.

3.7.6 Algorithms and Methods [NO]

The primary software methods/functions will be discussed in the following subsections. Each function is described as needed to provide a clear understanding of how each method achieves its goal. All applicable pseudo code can be found in Appendix II.

3.7.6.1 Microprocessor Functions [NO]

The function "setpins," will designate the pin variable names, and assign them based on whether they are an input or an output. See **Table 10** for more details. The function "sendPower_Devices," will send power to the devices. See **Table 11** for more details. The "setup" function, will do the basic setup for the program. This includes setting the pins, starting the serial port, starting the oneWire port, starting the I^2C bus, and configuring the timer for future use. See **Table 12** for more details. The function "tempSensor_Listen," will listen for temperature data. See **Table 13** for more details. The "airflowSensor_Listen" function will listen for airflow data, and return that data in character or string format. See **Table 14** for more details. The function "sendData2zigbee" will wait until the receiving ZigBee is ready to receive data, before transmitting the data collected. See **Table 15** for more details. The function "waitforConfirmation" returns Boolean corresponding to whether the microcontroller has received a confirmation from the data center. See **Table 16** for more details. The "dataModification" function will organize the data from all the sensors into one string. This will allow us to separate the data by a chosen character for easy decryption on the data center. See **Table 17** for more details. The "removePower_Devices" function will disconnect power from devices to begin sleep mode. See **Table 18** for more details. The "sleep" function will put the microcontroller into sleep mode. It will also start the timer, which will be used in the wake function. See **Table 19** for more details. The "wake" function will wake the device up when the timer is equal to the sample rate. The sample rate will be chosen depending on how often it is desired to collect data. See **Table 20** for more details. The function "isBatteryCharged" will get charge data from the charge controller, convert this data to a percentage of the total battery charge and return a Boolean depending on the charge level of battery. If charge is at 98% or greater, it will return a true to stop charging the battery, to prevent damage. See **Table 21** for more details.

Table 10. Functional Requirements to Set Pins.

Module	Set Pins
Inputs	None
Outputs	None
Functionality	Sets pins variable names and whether it is an input or an output pin.

Table 11. Functional Requirements to Send Power to Devices.

Module	Send Power to Devices
Inputs	None
Outputs	None
Functionality	Sends power to devices, by setting pins high.

Table 12. Functional Requirements for Setup.

Module	Setup
Inputs	None
Outputs	None
Functionality	Sets pins, starts the serial port, the oneWire port, the I^2C bus, and configures the timer for sleep.

Table 13. Functional Requirements to Listen for Temperature Sensor.

Module	Listen for the Temperature Sensor
Inputs	Data - from the Serial port
Outputs	Sensor data - in the form of a string
Functionality	Waits to receive data from the sensor, returns data as a string.

Table 14. Functional Requirements to Listen for Airflow Sensor.

Module	Listen for the Airflow Sensor
Inputs	Data - from the I^2C bus
Outputs	Data - in string format
Functionality	Waits to receive data from the I^2C bus, converts it to a string and then returns it.

Table 15. Functional Requirements for Transmission.

Module	Transmit collected data to the data collection

	center
Inputs	Organized Data - in string format
Outputs	Returns boolean - false if receiving ZigBee is busy
Functionality	Waits to receive data from the I^2C bus, converts it to a string and then returns it.

Table 16. Functional Requirements to Wait for Confirmation.

Module	Wait for Confirmation
Inputs	None
Outputs	Returns boolean - false if no errors happened
Functionality	Waits to receive confirmation from the data collection center, returns if error has happened.

Table 17. Functional Requirements for Data Modification.

Module	Data Modification
Inputs	Temperature Data, Airflow Data, Battery Data
Outputs	Returns string - combination of all data, with chosen character separating different data types
Functionality	Organizes data for ease of transmission and decryption at data center.

Table 18. Functional Requirements to Remove Device Power.

Module	Remove Device Power
Inputs	None
Outputs	None
Functionality	Sets Power pins low, to remove power from devices.

Table 19. Functional Requirements to Sleep.

Module	Microcontroller Sleep
Inputs	None
Outputs	None
Functionality	Activates sleep mode for the microcontroller, also starts timer which will be used to wake the device

	up.

Table 20. Functional Requirements to Wake.

Module	Microcontroller Wake-up
Inputs	None
Outputs	None
Functionality	Wakes up the microcontroller when the timer is equal to a given sample time.

Table 21. Functional Requirements for Battery Charge Indicator.

Module	Battery Charge Indicator
Inputs	Battery charge data from the charge controller
Outputs	Bool: True if battery is charged.
Functionality	Gets data from the charge controller and converts it to a percentage of total battery to check to see if charge level is at 98%. If so, it will return True.

3.7.6.2 Data Collection Center Functions [NO]

A library that was developed for communication between C# and ZigBee will be used. The "convertPacket_to_String" function will take a received packet from the communication bus and convert it to a string for easier use. See **Table 22** for more details. The "storeData" function will take a string in, and store that data into the database at a specific location. See **Table 23** for more details. The "getdatafromDataBase" function will return an array of data points that can be turned into a graph at a later time. See **Table 24** for more details. There are a few different libraries for C# to allow easy graphing. One of these will be used to stream line the process. For the "graphPoints" function, data will be graphed corresponding to data type; that is, Temperature vs. Time, Airflow vs. Time, Humidity vs. Time, and battery charge level. See **Table 25** for more details. Finally, the function "displayGraph" will update the previous graphs with the newly collected data added. The GUI will also display other information to the user, such as the total battery level. See **Table 26** for more details.

Table 22. Functional Requirements for Converting Packets to Strings.

Module	Convert Packet to String
Inputs	Received Packet - from ZigBee device

Outputs	Returns a string of data
Functionality	Takes a received packet from the ZigBee device and breaks the bytes down into usable strings.

Table 23. Functional Requirements for Storing Data.

Module	Store Data
Inputs	A string containing a specific set of data; temperature, airflow, or battery.
Outputs	Outputs - to a local database
Functionality	Stores a specific set of data in the data base. Moves entry location to the next spot.

Table 24. Functional Requirements for Getting Data from Database.

Module	Get Data From Database
Inputs	None
Outputs	3D Array
Functionality	Array will contain X, Y coordinate along with data type.

Table 25. Functional Requirements for Graphing Data.

Module	Graph Data
Inputs	Data type, Data Points
Outputs	Creates a Graph File
Functionality	Using data type and data points, creates and adds points to a graph file. This file will later be called to be displayed.

Table 26. Functional Requirements for Displaying Graph.

Module	Display Graph
Inputs	None.

Outputs	None.
Functionality	Updates Graphical User Interface with new graphs and information.

3.8 Final Design [TG RG]

Figure 23 shows the final block diagram of the proposed HVAC Monitoring Device, while **Figure 24** shows the power consumption profile generated based upon the component selections.

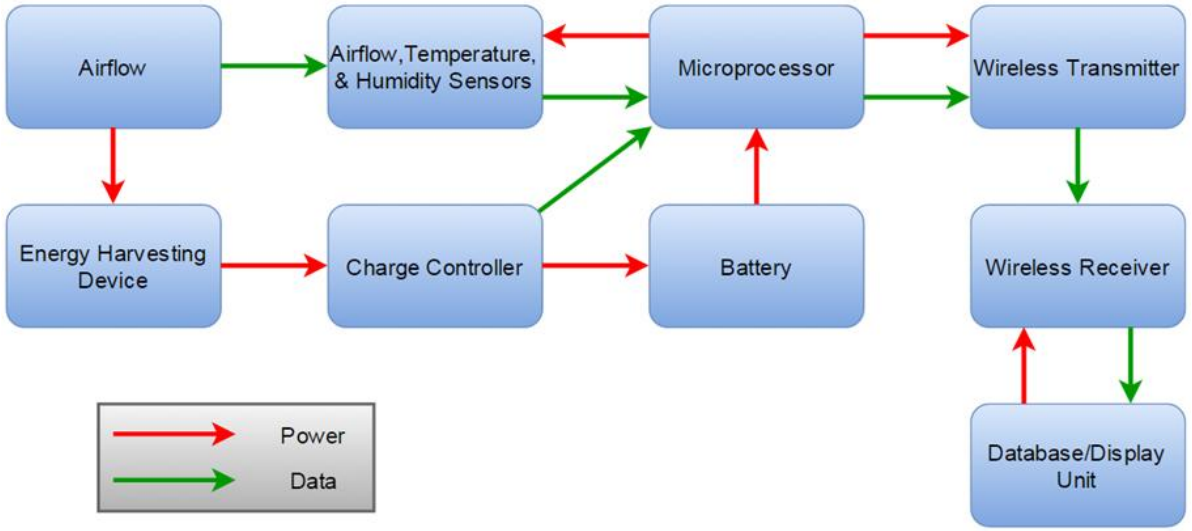


Figure 23. Final Design Block Diagram.

Figure 23 illustrates the flow of data and power after implementation of final design constraints. By using the Savonius wind turbine to generate power, the energy harvesting apparatus will continually generate power while the airflow is present in the air duct. The backpack and battery together provide an internal regulation and stable voltage range for the unit to operate safely and comfortably provide voltage to the transmission and data collection system. The temperature and airflow sensors are placed in the duct in such a way that the first point of contact for air in the duct is the sensors. This way, the sensors will be able to provide data that is not manipulated by the harvesting device or and change in airflow in the system.

The microprocessor, housed inside the energy-harvesting device, controls the wireless and autonomous data acquisition and transmission. Using the wireless transmitter and receiver, the data can easily be manipulated to transmit to the database and displayed

seamlessly with the user interface. The user interface is required to allow for the database to present the information in a manner that allows for the system characteristics to be easily compared.

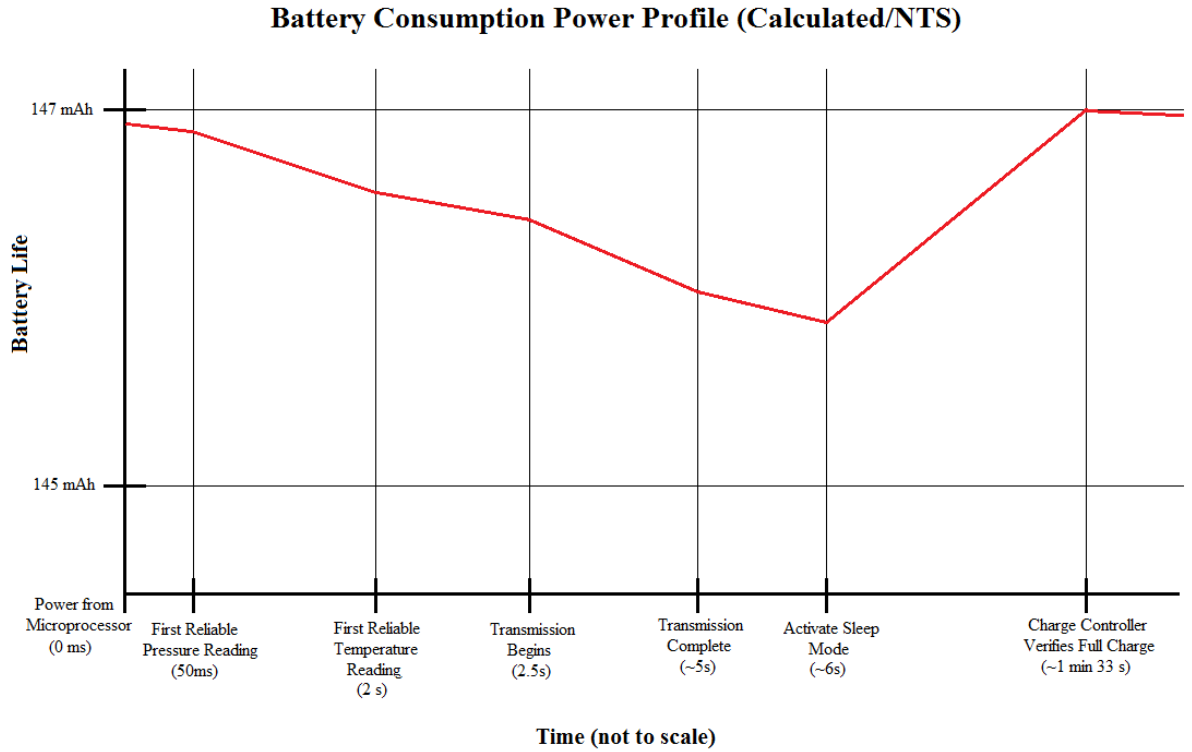


Figure 24. Power Profile Graph.

The above power profile outlines the intended operation of the life cycle of the proposed HVAC monitoring device. The wake cycle consists of 4ms wake-up for the microprocessor, 15ms wake-up for the ZigBee transmission device, 50ms wake-up for the pressure sensor, and 2 full seconds for the first reliable temperature reading. Throughout the course of this wake-up process, battery power will be drained at approximately 100mA per hour.

Once transmission begins, the sensors will no longer be pulling any current. However, the XB24-Z7WIT-004 transmission device pulls a maximum of 40mA as it transmits. The power profile above was calculated assuming that the transmitter would pull 40mA throughout the entire transmission process. This process is estimated to take less than two seconds, according to the Adafruit website [20]. The device will resume sleep mode once a transmission verification has been received.

The wake-up cycle does not begin at a fully charged battery, because the charge controller holds the charge at 147 mAh (98%). The microcontroller requires 45nA in its sleep mode, but this is negligible since a trickle charge from the charge controller will constantly be keeping the battery topped off. Assuming the wake/transmission cycle takes 6 seconds, it has been calculated that it will take the battery 1 minute 33.6s to recharge.

The proposed final schematic for the device is shown in **Figure 25**.

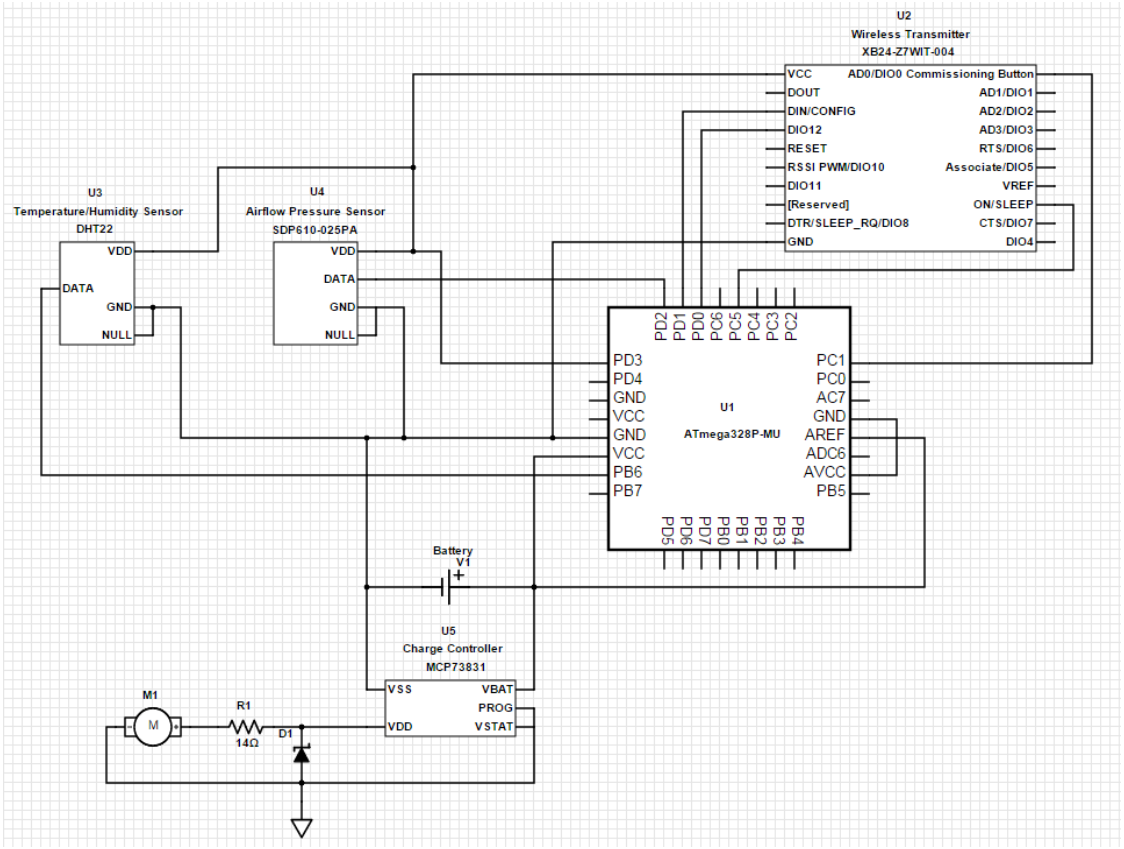


Figure 25. Final Design Device Schematic.

The final version of the schematic that was implemented in the prototype is shown in **Figure 26**.

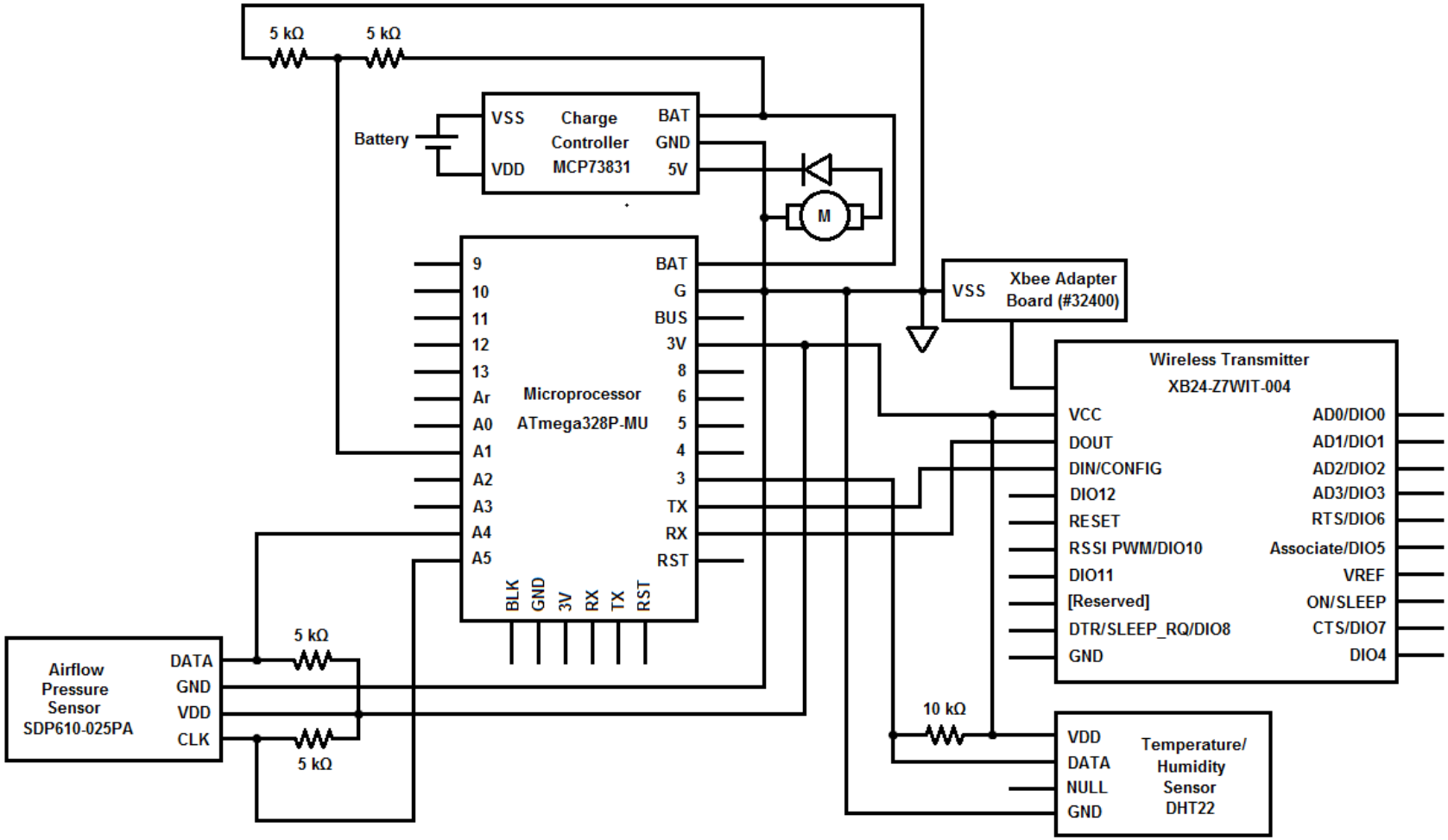


Figure 26. Prototype Final Schematic.

As can be seen by comparing the initial design and the final product, there were only a few changes to the schematic that was made in the process of implementation. Battery level was changed from being read straight from the battery to requiring a voltage divider to work with the analog input on the microprocessor, which was added to the circuit. Other resistors that were added were pull-ups for voltage signals that were needed for the microprocessor to read them. In addition, a diode was added to the input of the generator to stop battery voltage from passing along this path. This was not an expected outcome, but was easily rectified once discovered.

4. Operation Instructions

4.1 Hardware Operation [TG]

The HVAC monitoring turbine has two primary components:

1) The turbine/housing unit
2) The sensor pad

Each of these components must be mounted in the HVAC unit at the exit of the blower. In order to guarantee accurate insight into the performance of the unit, the sensor pad must be placed low in the vent, as close to the blower as possible. The turbine is to be placed along the wall which has most access to airflow in the unit.

First, the user must ensure that the location of placement for the turbine has sufficient airflow to power the device. With the blower on, the turbine should begin rotating on its own once it is placed in the vent. If the turbine does not begin rotating within 15 seconds, move the turbine closer to the blower, if possible. Be sure that the turbine rotor is not resting against any screws or protrusions which may be in the vent.

To mount the turbine to the wall, peel the adhesive away from the Velcro which is connected to the back of the turbine. Press the turbine against the wall for 20 seconds, then let go. Do not un-velcro the device for 24 hours after installation, as the adhesive must sit to ensure stability.

With the turbine mounted, place the sensor pad as low in the HVAC blower unit as possible. The sensors should sit approximately one to two feet closer to the blower than the turbine, so the airflow is uninterrupted where the readings are being taken. Remove the adhesive from the back of the Velcro on the sensor pad and press against the wall for 20 seconds, then let go. Do not un-velcro the pad for 24 hours after installation, as the adhesive must sit to ensure stability.

The device is powered on automatically when the turbine spins and provides sufficient power to the electronic components, so once it is set in the vent, there should be no required maintenance.

4.2 Software Operation [TG]

The software for the HVAC monitoring device is very straightforward and simple to use. Once it has been installed, the user only has to run the executable file and press “start” at the top of the screen. This is the only necessary step for the HVAC Monitoring Device to be fully functional. The device will sample the air in the unit once every minute, until the “stop” button is pressed.

5. Testing Procedures

5.1 Turbine

5.1.1 Power Testing [TG]

Testing of the turbine was done on a 4-ton, 150 CFM/ton system, to ensure that it would function even on the weakest HVAC units. The testing was done at the exit of the blower mouth, where the unit would be placed in an actual application. **Figure 27** shows a graph which was generated over the course of a 5 minute testing period, with each sample being taken 15 seconds apart.

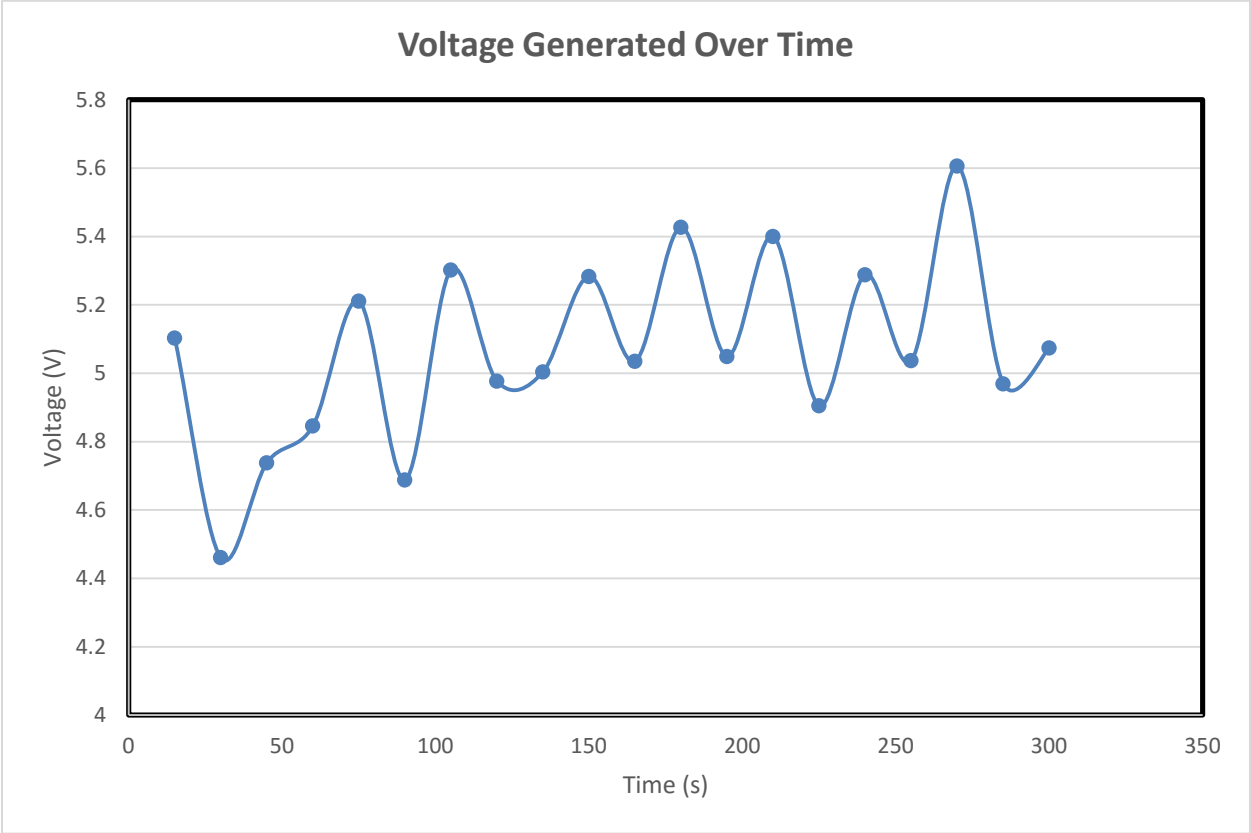


Figure 27. Voltage Generated Over Time.

The voltage generated consistently exceeded our power requirement of 3.7V, even with the blower running at its lowest setting. This confirms that the device will be able to constantly charge the battery throughout its operating life.

5.1.2. Airflow Disruption [TG]

Another concern for the turbine was the effect it would have on the overall function of the system. In order to verify that the device would not impede airflow to the degree that it affects the functionality of the system, samples of airflow were taken at various

points in the exit of the blower using an airflow meter. **Figure 28** shows where the 40 samples in the airstream were collected, as well as the wand with which they were measured.

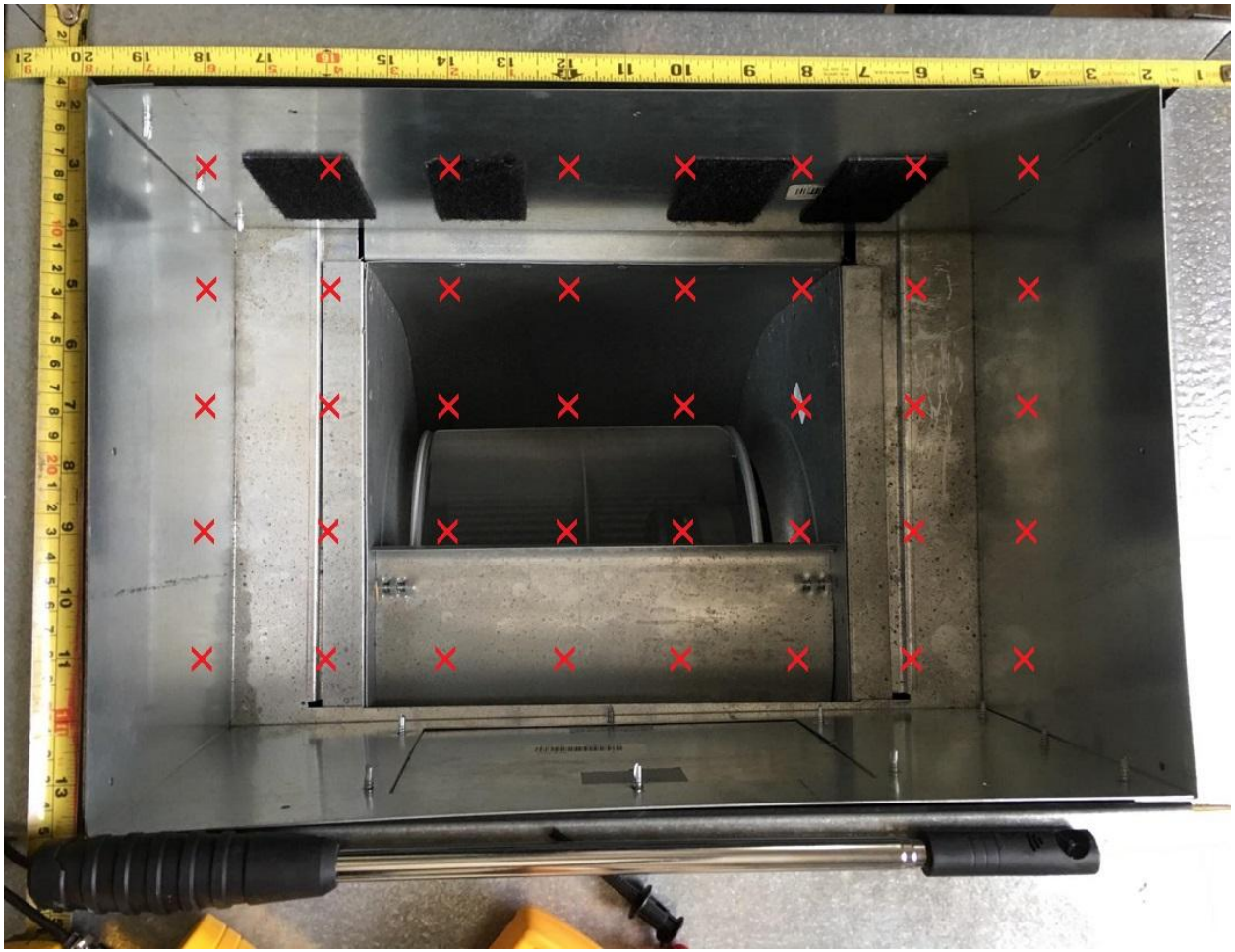


Figure 28. Sample Points for Airflow Data Collection.

The data collected is shown in **Table 27**, **Table 28**, and **Table 29**. **Table 27** was generated by measuring the airflow in the vent without the turbine in place. **Table 28** was with the turbine in the vent, and **Table 29** is the difference between the two.

Table 27. Airflow without Turbine.

1.4	5.6	8.6	11	10.6	5.8	6	2.3
1.3	6.2	8	11.1	8.1	5.9	3.4	1.4
1	5	7.6	9.6	3.8	3.4	2.4	1.5
1.2	2.1	6.3	9.9	2	1.5	1.5	2.1
1.3	3.3	9.1	8.6	2.1	1.1	1.5	2.5

Table 28. Airflow with Turbine.

1.2	5.1	6.5	6.5	5.1	1.6	4	3.9
1.6	2.3	2.3	1.8	3.5	1.5	2.3	1.9
0.7	1.1	2.1	3.5	1.1	1.4	1.5	2.2
1.9	3	5.9	9.2	4.9	3.2	2.8	2.5
4	4.6	7.3	10.1	9.6	7.8	6.3	4.9

Table 29. Difference in Airflow.

-0.2	-0.5	-2.1	-4.5	-5.5	-4.2	-2	1.6
0.3	-3.9	-5.7	-9.3	-4.6	-4.4	-1.1	0.5
-0.3	-3.9	-5.5	-6.1	-2.7	-2	-0.9	0.7
0.7	0.9	-0.4	-0.7	2.9	1.7	1.3	0.4
2.7	1.3	-1.8	1.5	7.5	6.7	4.8	2.4

Figure 29 shows a contour map of the airflow at the exit of the HVAC unit with the turbine, while **Figure 30** shows a map of the airflow without the turbine. The data in each chart has been normalized, for easy comparison. The colors range from dark blue, being 0 CFM of airflow, to dark red, which represents the maximum value of airflow that was collected, 11.1 CFM. The Matlab code which was used to generate the contour maps is shown in **Appendix III**.

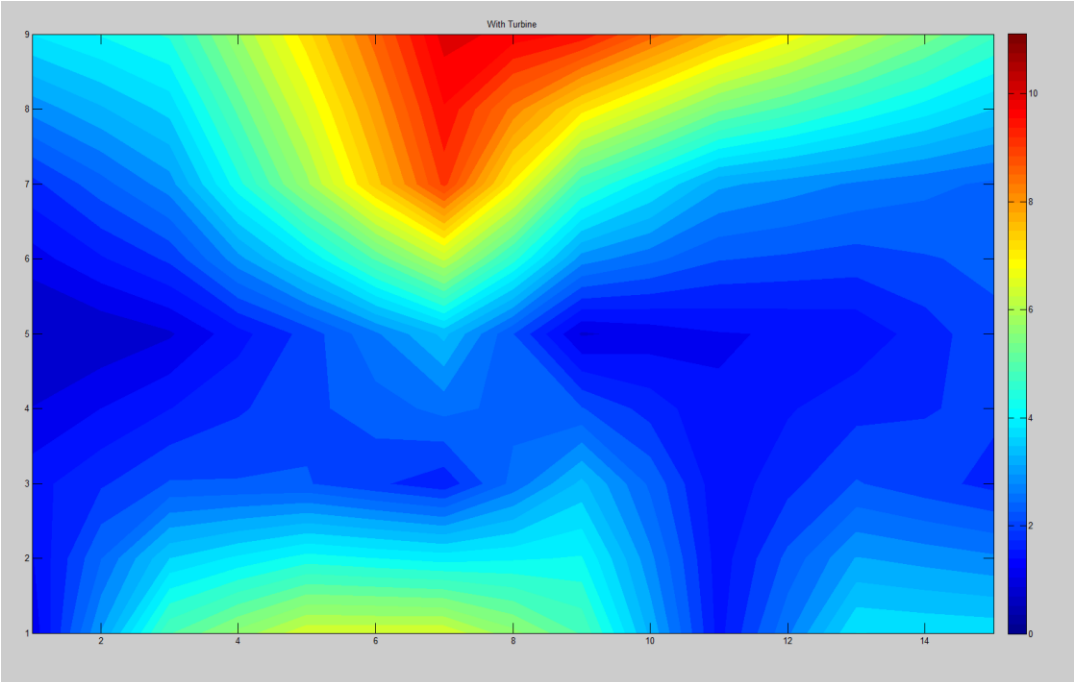


Figure 29. Airflow Contour Map with Turbine (CFM).

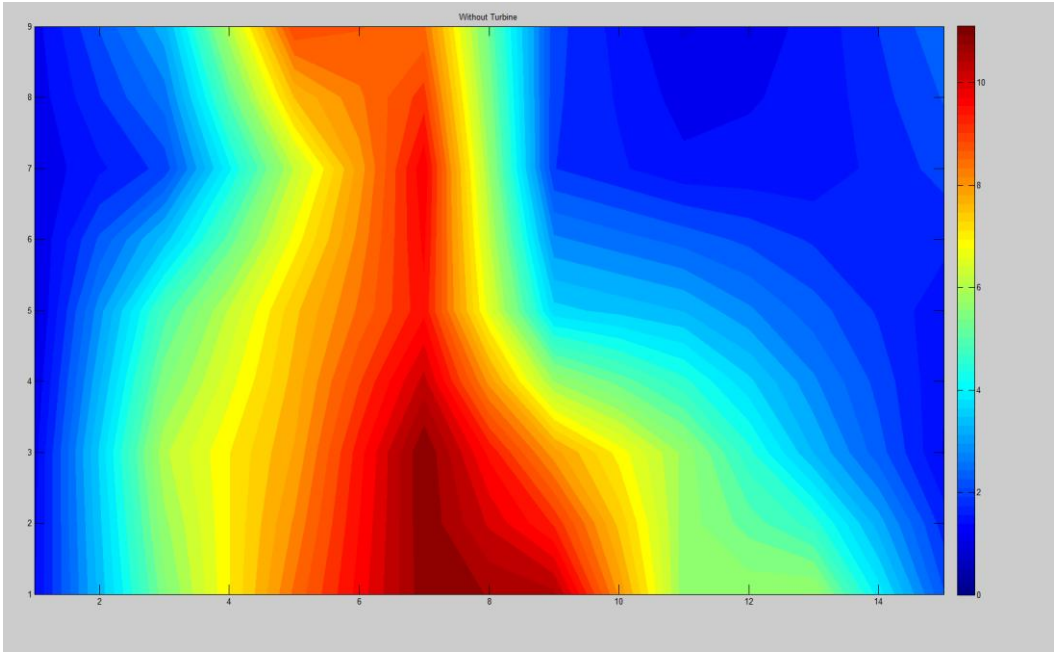


Figure 30. Airflow Contour Map without Turbine (CFM).

The darkened area of **Figure 31** represents the area of airflow most blocked by the turbine. There was an increase in airflow in the part of the vent which did not have the turbine, which shows that the airflow moved around the turbine if it was not being caught and converted to electrical power.

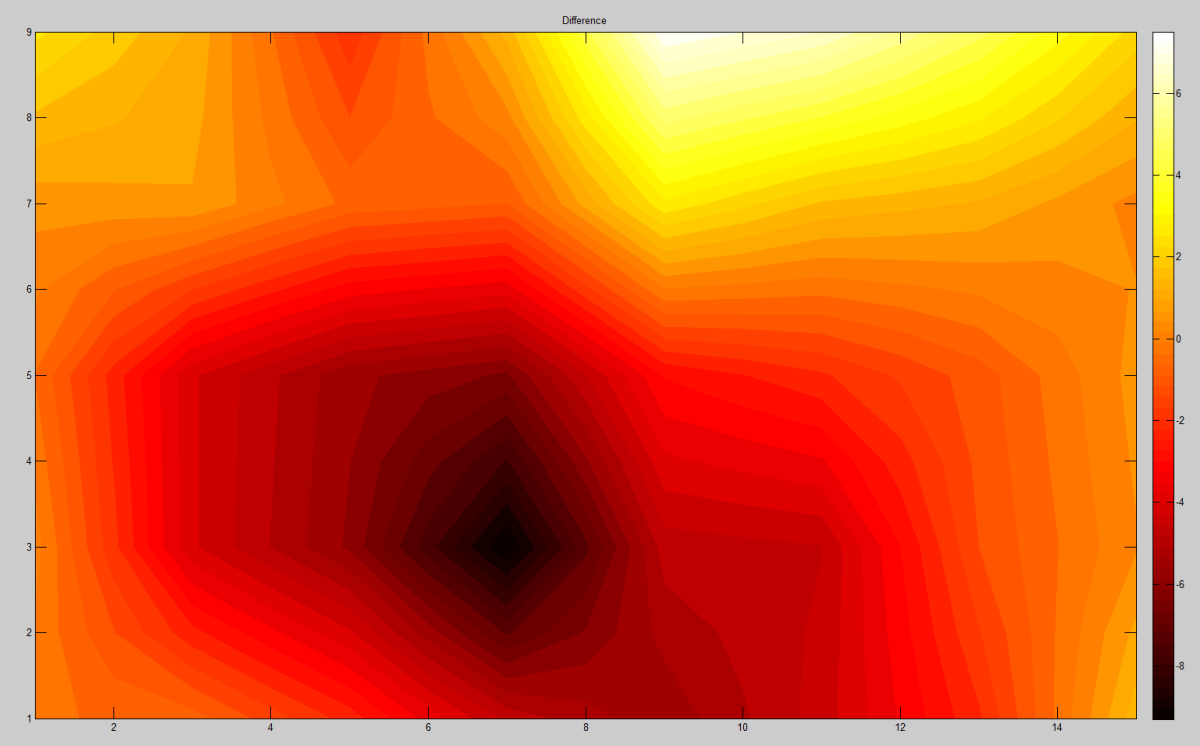


Figure 31. Contour Map of Difference in Airflow (CFM).

The contour maps provide a visual confirmation that airflow from the blower is mostly redirected around the turbine, if it is not used as power within the monitoring system. The mean airflow for each 2x2 inch^2 area was found to be 4.841 CFM without the turbine in place, and 3.711 CFM with the turbine in the vent. The median difference in airflow came out to be -0.65 CFM. Although the turbine does use some of the airflow to generate power, even in the smallest 150 CFM system these differences in airflow will not be noticeable to the occupants of the space the HVAC unit is heating or cooling.

5.1.3 Conclusions [TG]

The turbine component of the HVAC Monitoring Device successfully meets all marketing and design requirements set out for the project. Generating the contour plots was difficult because the airstream was highly inconsistent. However, the numbers recorded were representative of the airflow at each point, and care was taken to ensure that the data collected was as accurate as possible.

Testing was only performed on the one HVAC unit that was available for this project. However, it was a 150 CFM unit, and more airflow would only mean more power to the device. For the purposes of this project, testing with this HVAC unit was sufficient to prove that the concept of a self-powered airflow monitoring system in an HVAC application is possible, and that one can expect similar results in larger systems.

5.2 Battery and Charge Controller [TM]

In order to test components prior to total integration, the battery and charge controller/backpack were tested for capabilities and datasheet values. Testing the capabilities of the battery required checking the ability of the battery to continuously power all the processes in the system.

The charge controller schematic shown in **Figure 32** shows the immediate layout of the backpack charge controller. This was used to solder the connections between the input power from the generator as well as the output power cable to the rest of the hardware.

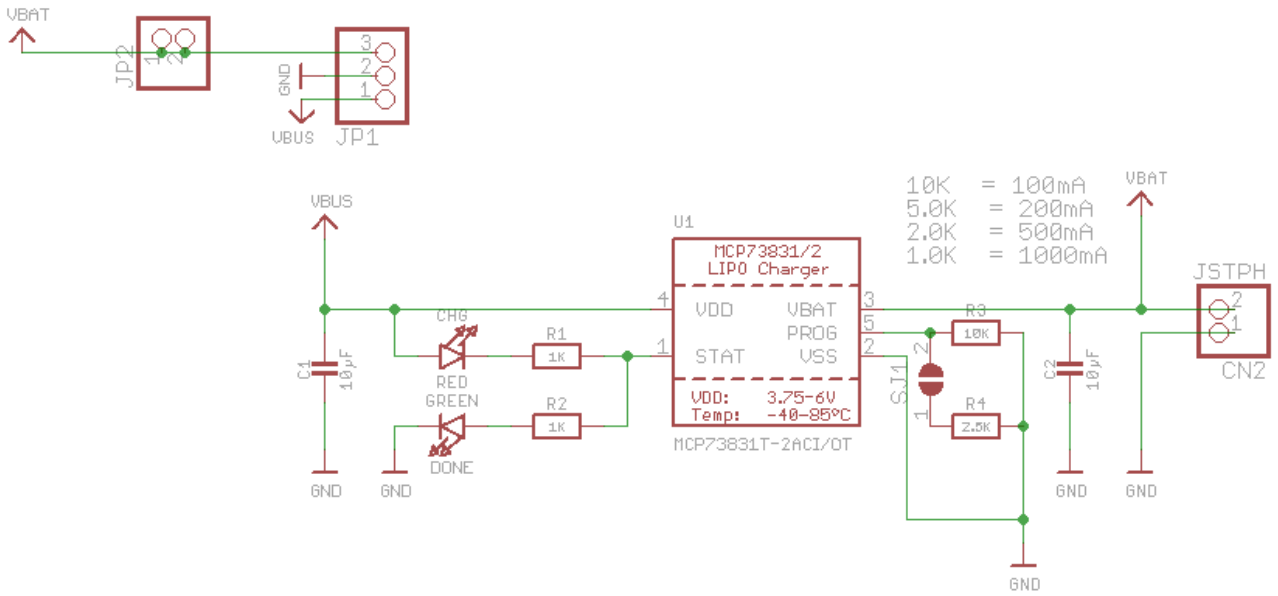


Figure 32. Charge Controller Schematic.

Figure 33 shows the physical layout of the charge controller. This charge controller was soldered on directly to the microprocessor, saving space in the process. This configuration allowed for the power to be directly input into the microprocessor from the battery, after going through the charge controller. By wiring up the 5V and GND terminals, the input power from the generator (approximately 4.5-6V at 0.1A, or .45-6W) was able to be put into the charge controller, and subsequently the battery, allowing constant charge when necessary.

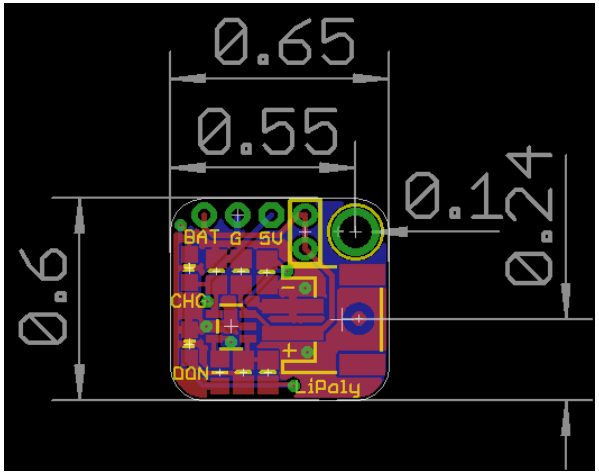


Figure 33. Physical Layout of Charge Controller.

One of the largest selling points for this design is complete autonomy. This cannot be achieved if the battery needs to be constantly changed due to damage or lack of power. By testing the battery and charge controller in the system with a higher input voltage than necessary, the power subsystem can be shown to confidently deliver usable power and keep the microprocessor at a proper voltage in order to maintain functionality. By placing a 5.5V DC voltage onto the battery/charge controller system, the power subsystem is being tested in multiple ways. Firstly, 5.5V is above the battery operating voltage, so the charge controller has to regulate that down to a usable voltage, in this case 4.25V. The charge controller model used for the power subsystem regulates the voltage across it at this voltage.

A concern with this voltage, however, was overheating the system internally. By overheating the system internally, the temperature sensor would not be able to accurately accept meaningful data. By regulating the voltage down to a usable level, there will be heat created from the extra voltage. By placing the turbine in a suitable position in the duct, the power subsystem is able to safely generate between 4.5V-6V. Previous testing yielded voltages upwards of 15V, which was well outside of the safe range of parameters of the system. By moving the turbine subsystem slightly farther away from the point of generation, the power subsystem is able to safely generate a usable voltage. The result of this strategic positioning can be seen in **Figure 34**.

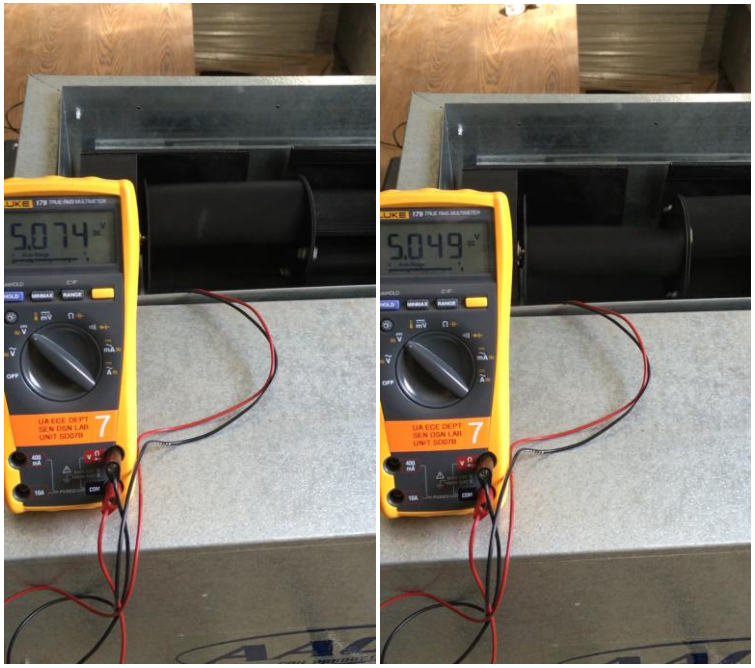


Figure 34. Voltage Generation Testing in Duct.

Another test that was done multiple times throughout the process was the integration of the battery and charge controller with the microprocessor. The data gathered from the sensors would be meaningless if it could not be transmitted to a bay central unit for data processing and storage. Slight problems occurred throughout the process due to power from the battery alone not being able to operate the microprocessor correctly. This problem was discovered as having an open ground due to unclean initial wiring, but this was eventually discovered and resolved. An example picture of this test being performed can be seen in **Figure 35**.

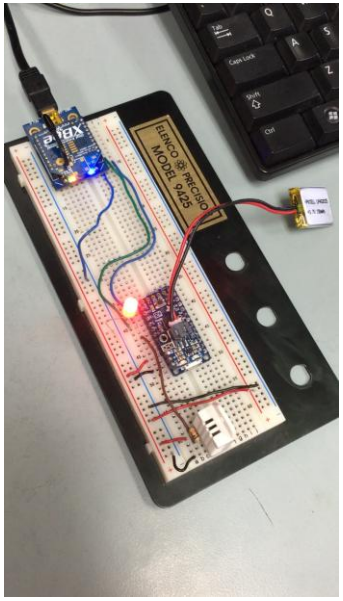


Figure 35. Communications Testing Using Battery Power.

After full integration, the system showed signs of back feeding the motor/generator. By having the battery connected directly from the generator, power would flow backwards through the circuit and power the motor, instead of the other way around. This would cause the airflow to fight the generator rather than freely allowing it to rotate. After evaluating the problem, a diode was placed between the generator and the charging apparatus in order to have current flow become one linear path, rather than two conflicting paths. The diode placed was rated for 600V and 1A, meaning our maximum generated voltage and current from the generator are well below rated values, causing the diode to remain cool and thus not overheat in the system. The diode used can be seen in **Figure 36**.

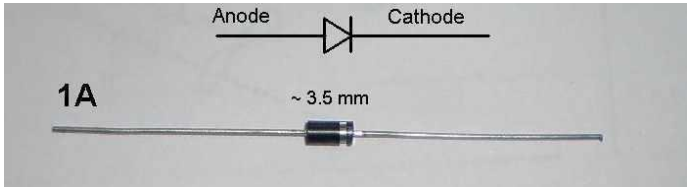


Figure 36. 600V 1A Rated Diode Configuration.

Finally, with full system integration, the power system was designed to show a change in battery level, both gaining power from the generator and using power to transmit data. Using a transmission rate of two seconds per transmission, the test showed a usage of power through voltage level on the battery. The GUI was developed in order to show both battery level and progression of the battery level. In the data table, the output row will show the remaining voltage level of the battery in the final column.

5.3 Wireless Communication [NO]

Wireless communication was a large part of this project and the testing aspect of each subsystem. The development board that housed the microprocessor did not allow for serial communication over USB directly from the device. For testing and debugging purposes, it was necessary to communicate wirelessly with the microprocessor continuously.

Testing of the wireless communication system was done on a computer running XCTU, a program designed for serial communication. For testing to begin, each ZigBee had to be configured properly using this program. First, the firmware of each ZigBee was changed from ZigBee Router API to ZigBee Router AT. Next, a PAN ID was selected to be 325 for the purposes of this project, as seen in **Figure 37**.

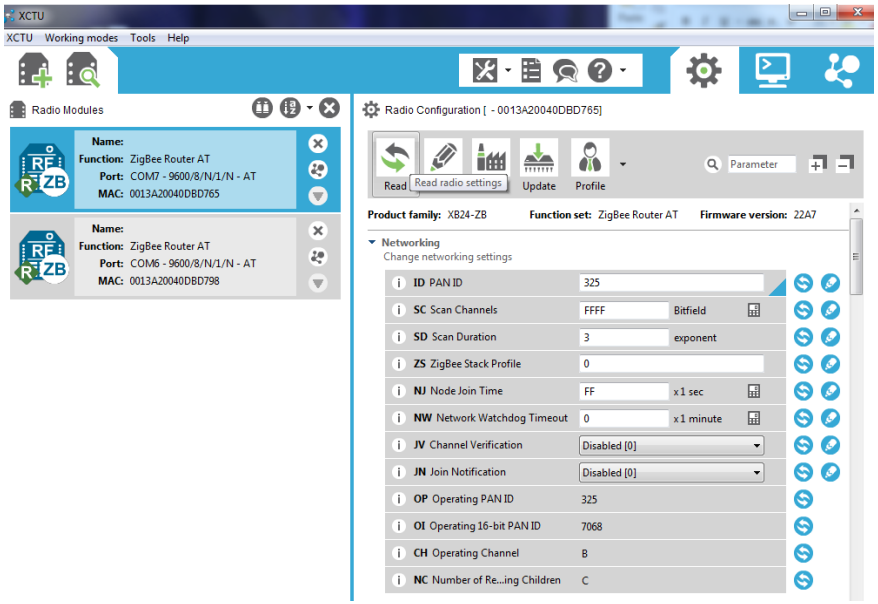


Figure 37. XCTU Used to Configure ZigBees.

Next, each of the destination addresses needed to be set. Since this system uses point to point communication, the other ZigBee's MAC address provides the destination address, as seen in **Figure 38**. The MAC address is split between the high and low destination addresses.

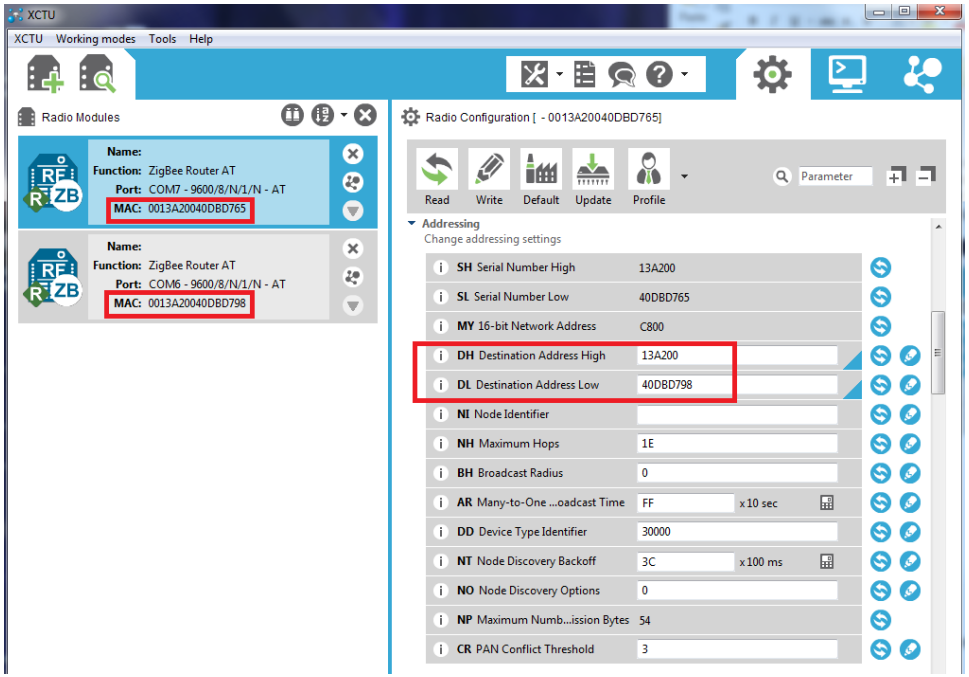


Figure 38. Configuring ZigBee Destination Addresses.

With both ZigBee's configured, the console mode was used in XCTU to test communication between the two modules. In order to achieve this, one must open communication with each ZigBee, as seen in **Figure 39**.

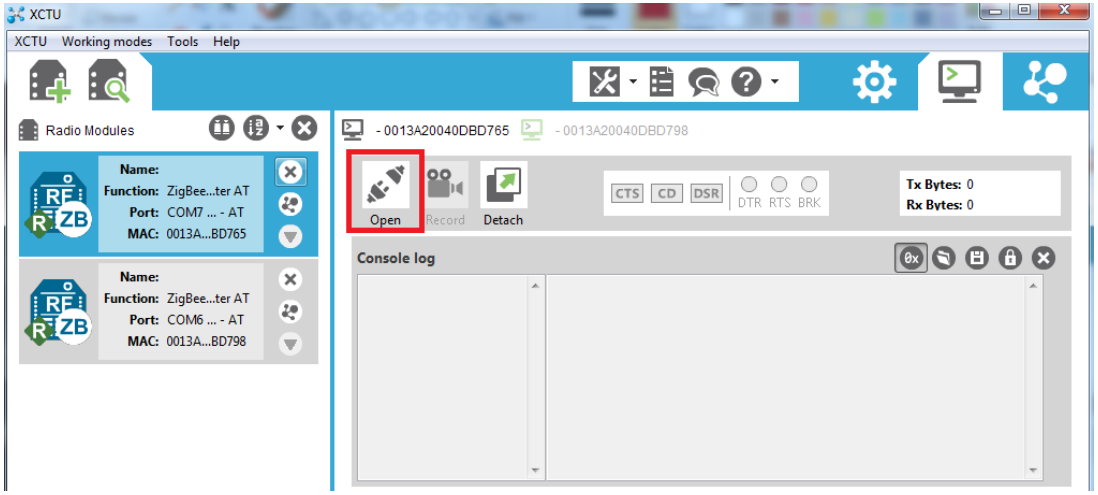


Figure 39. Opening Zigbee Communications.

With open communications with both Zigbees, the system was ready to communicate between the two devices. This means that typing in the console log text box will send serial data to the other ZigBee wirelessly. Each Zigbee's serial data will be color coded to display whether it is a received or sent data. **Figure 40** displays these wireless communications in progress.

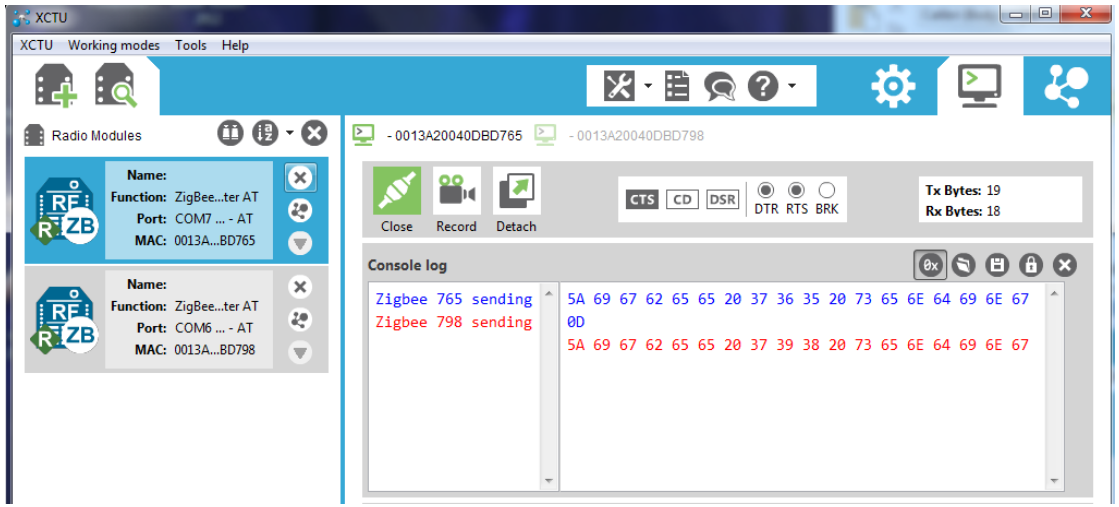


Figure 40. Sending and Receiving Zigbee Data.

With both sides communicating with each other, the ZigBee was wired to the ProTrinket using a solderless breadboard. The connections were the following: ZigBee pin DataIN to the TX pin on the ProTrinket, ZigBee pin DataOut to the RX pin on the ProTrinket, ZigBee GND to the ProTrinket GND, and ZigBee 3.3V Pin to the 3.3V pin on the ProTrinket. With both devices wired together, the connection from the transmitting ZigBee to the computer was removed and the receiving ZigBee was left connected to the computer. With simple code running on the microprocessor, communication between the ZigBees on the XCTU console can be seen. Code used for this purpose and the resulting output can be seen in **Figure 41**.

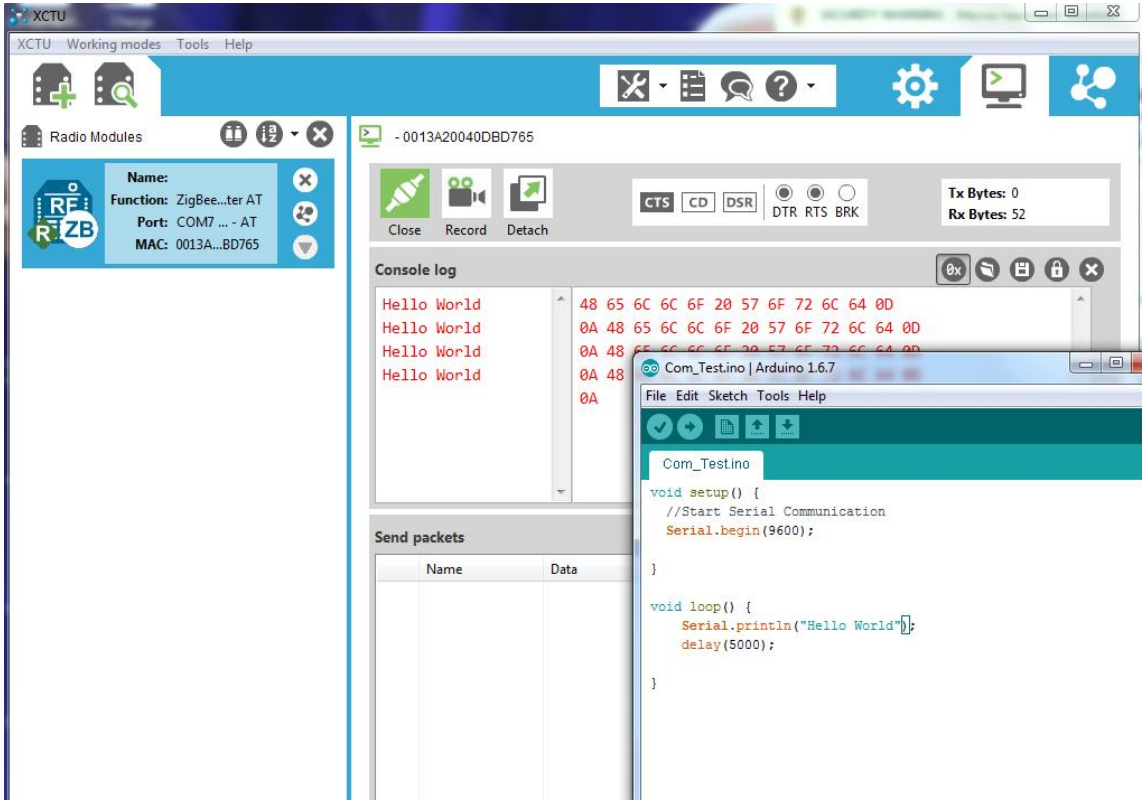


Figure 41. Receiving Data from Microprocessor.

5.4 Temperature Sensor [NO RG]

The temperature and humidity sensor had some demo code distributed with it, so after wiring the temperature sensor to the microprocessor, this code was used to test the functionality of the sensor. With the code running on the microprocessor and the ZigBee attached, the output of the sensor using the console in XCTU can be seen. This output can be seen in **Figure 42** and the corresponding code can be found in **Appendix IV**.

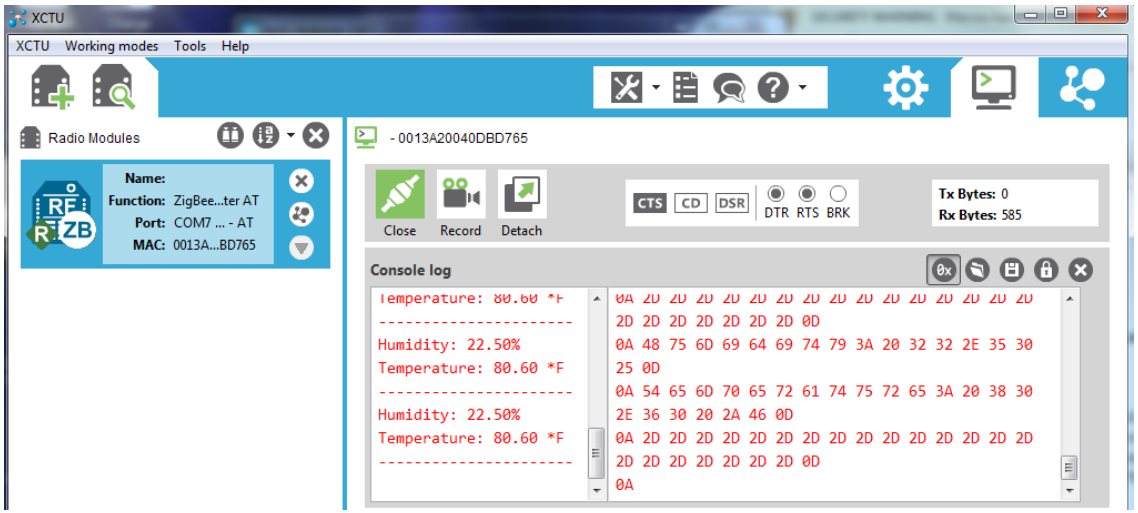


Figure 42. Temperature and Humidity Sensor Test.

These results could then be shown to be true by adjusting the temperature and humidity near the sensor. To show that the temperature was correct, a personal hair dryer was used to simulate hot or cold air blowing across the sensor. For humidity, breathing on the sensor proved to be one of the best ways to show that it was responsive. Very quickly, the readings adjusted appropriately for all of these cases, proving that the sensor was functioning properly.

5.5 Pressure Sensor [NO RG]

The first step to get the pressure sensor working was to identify the I^2C address of the device. First, the pressure sensor was connected to the I^2C bus of our microprocessor. Then, a I^2C_scanner program was used, which ran on the microprocessor and

displayed the address of the device. It was determined that the address of the pressure sensor was 0x40, or 64. This can be seen in **Figure 43**, with code available for reference in **Appendix IV**.

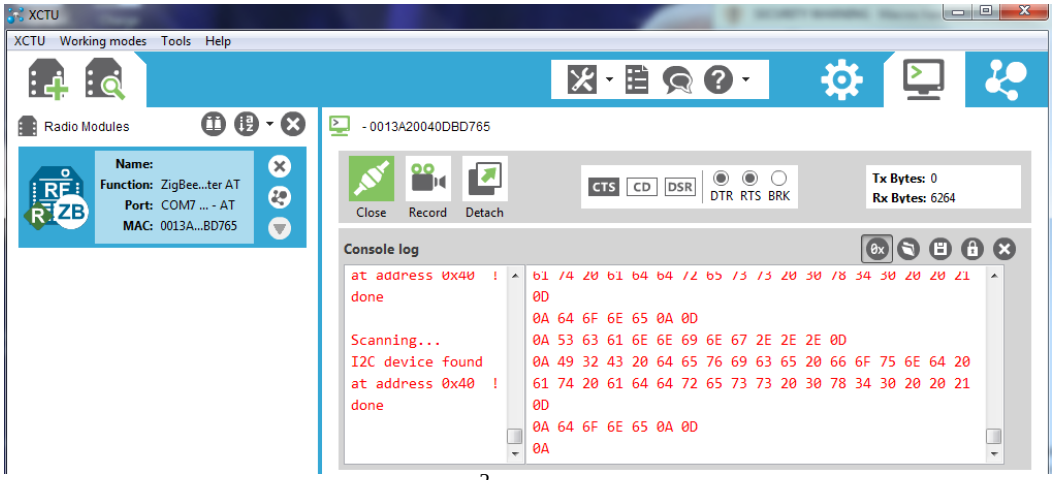


Figure 43. I^2C Device Scanner.

After finding the address of this sensor, the next step was to test it using test code supplied by the manufacturer. The output of this code can be found in **Figure 44**.

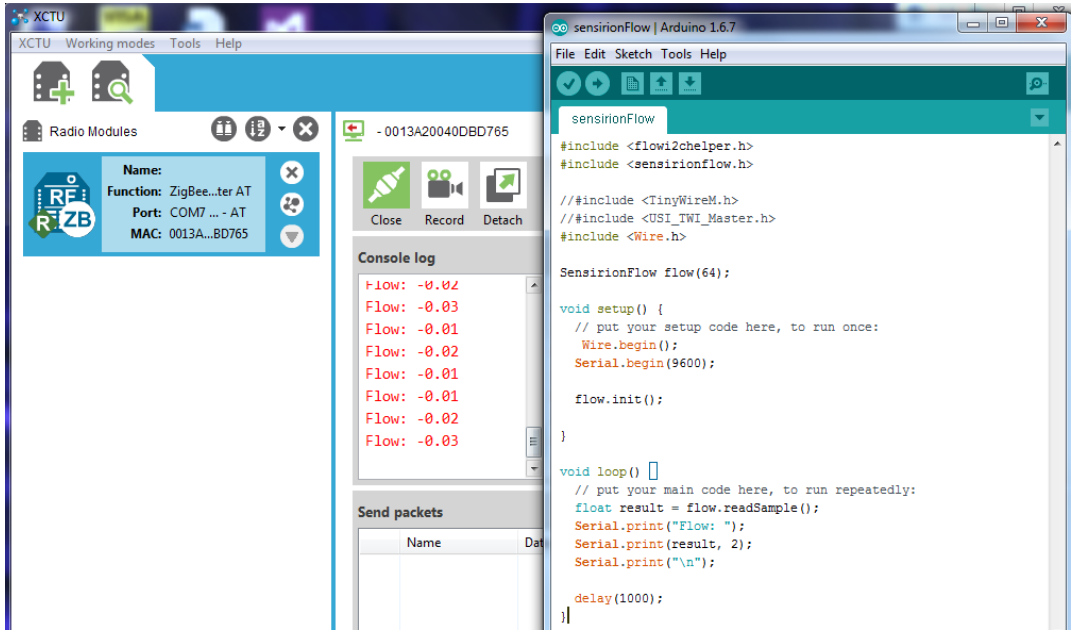


Figure 44. Pressure Sensor Test.

This output was in terms of Pascals of pressure, which then had to be converted to a reading of airflow. This was accomplished using the mathematical equation shown in **Figure 16**. The personal hair dryer was once again used for this sensor to show that as different amounts and directions of airflow were put across the ports, the sensor readings changed accordingly. This showed that in one direction, the readings were a positive number, and were negative in the opposite direction.

5.6 GUI [NO]

For the GUI to communicate with the data collection center, it was necessary to be able to communicate with the ZigBee attached to the computer. This was done using a Windows Form Project with a serial bus to communicate with the ZigBee, in order to display serial data from the microprocessor. Next, all the data being sent from the microprocessor was organized in a fashion which could easily be parsed on the GUI side. The data being sent from the microprocessor was considered a string type, therefore after parsing the data into temperature, humidity, airflow, and battery data, the string needed to be converted to a floating point number. These floating point numbers served as a starting point for the GUI to perform needed operations on them. These operations ranged from math for the pressure data to graphing data points. The final GUI format can be seen in **Figure 45**.

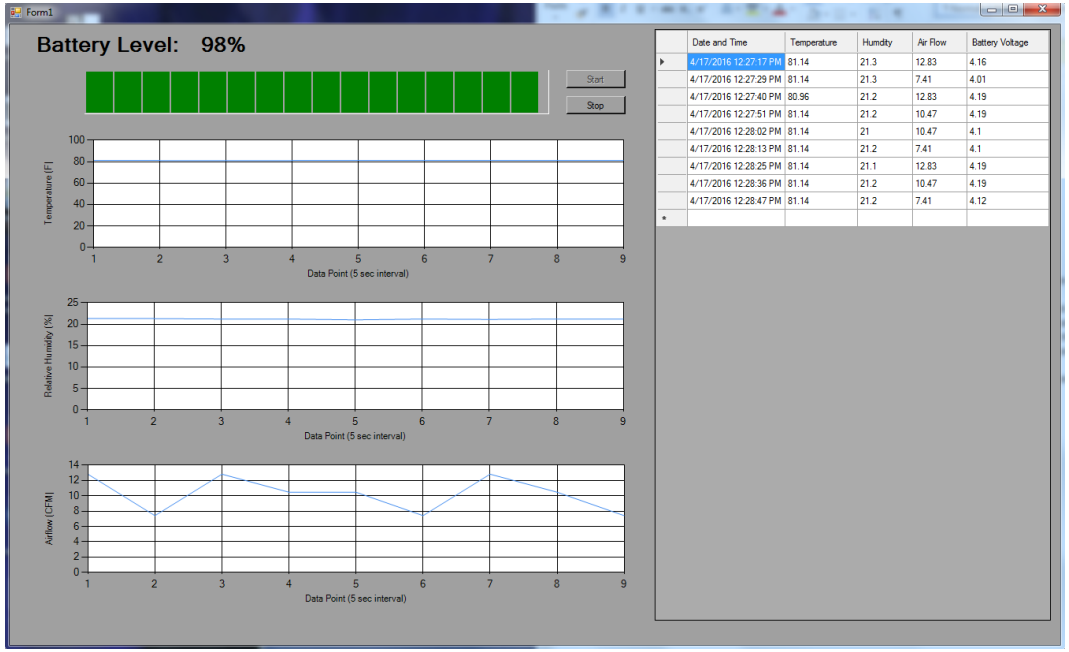


Figure 45. Final GUI.

6. Parts List

Table 30. Complete Part List for the Accepted Technical Design.

Qty.	Description	Suggested Vendor	Vendor Part Num.	Catalog #/Page #/Website	Unit Cost	Total Cost
1	DC Motor	Ebay	N/A	http://www.ebay.com/itm/12V-DC-160rpm-Permanent-Magnetic-Planet-Gear-Motor-New-M32-/261	34.99	34.99
1	Battery	Adafruit	1317	http://www.adafruit.com/products/1317	5.95	5.95
1	Charge Controller	Adafruit	2124	https://www.adafruit.com/products/2124	4.95	4.95
1	Airflow Sensor	Mouser Electronics	403-SDP610-025PA	http://www.mouser.com/ProductDetail/Sensirion/SDP610-025PA/?qs=%2fha2pyFadugv7nJxTh9K	35	35.00
1	Temperature/Humidity Sensor	Adafruit	385	http://www.adafruit.com/products/385	9.95	9.95
1	Microprocessor	Adafruit	2010	http://www.adafruit.com/products/2010	9.95	9.95
2	Wireless Transmitter	DigiKey	968	http://www.digikey.com/product-search/en?vendor=0&keywords=XB24-Z7WIT-004	17	34.00
1	Zener Diode	Greg Lewis	N/A	5.1V Zener Diode, power rating 2550 mW	0	-
1	Resistor	Greg Lewis	N/A	14 ohm resistor, 3450 mW	0	-
					Total	**$134.79**

Table 31. Final Parts List for Prototype.

Qty.	Description	Suggested Vendor	Vendor Part Num.	Catalog #/Page #/Website	Unit Cost	Total Cost
1	DC Motor	Ebay	N/A	http://www.ebay.com/itm/12V-DC-160rpm-Permanent-Magnetic-Planet-Gear-Motor-New-M32-/261163449627	34.99	34.99
1	Battery	Adafruit	1317	http://www.adafruit.com/products/1317	5.95	5.95
1	Charge Controller	Adafruit	2124	https://www.adafruit.com/products/2124	4.95	4.95
1	Airflow Sensor	Mouser Electronics	403-SDP610-025PA	http://www.mouser.com/ProductDetail/Sensirion/SDP610-025PA/	35	35.00
1	Temperature/Humidity Sensor	Adafruit	385	http://www.adafruit.com/products/385	9.95	9.95
1	Microprocessor	Adafruit	2010	http://www.adafruit.com/products/2010	9.95	9.95
2	Wireless Transmitter	DigiKey	968	http://www.digikey.com/product-search/en?vendor=0&keywords=XB24-Z7WIT-004	17	34.00
1	Zener Diode	Greg Lewis	1N4003	200V Zener Diode, 1A	0	-
2	XBee Explorer USB	Sparkfun	WRL-11812	https://www.sparkfun.com/products/11812	24.95	49.90
1	2mm Pitch Prototype Board	Digikey	V2013-ND	http://www.digikey.com/product-detail/en/vector-electronics/8017/V2013-ND/417271	17.53	17.53
1	.1in Pitch Prototype Board	Greg Lewis	N/A	Standard Prototype Board with .1 inch pitch	0	-
4	5kΩ Resistor	Greg Lewis	N/A	Carbon Film, 1/4 W, 5% Resistor	0	-
1	10kΩ Resistor	Greg Lewis	N/A	Carbon Film, 1/4 W, 5% Resistor	0	-
					Total	**$202.22**

7. Design Team Information

Ryan Gerhart, Electrical Engineering, Project Leader
TJ Ghinder, Electrical Engineering, Archivist
Tyler Miller, Electrical Engineering, Hardware Manager
Nick Owens, Computer Engineering, Software Manager
Dr. De Abreu-Garcia, Faculty Advisor

8. Conclusions and Recommendations [TM RG]

8.1 Satisfying the Design Requirements

Table 33 and Table 34 below further illustrate how the design meets the marketing and engineering requirements, respectively.

Table 32: HVAC Monitoring System Marketing Requirements

Marketing Requirements	Implementation Satisfying Requirement
Device must be self-powered to incur no additional cost to run the HVAC system.	Using the airflow and energy harvesting system, the system will use the battery to power all internal functions with no additional cost to the HVAC system.
Device must not require constant or continuous maintenance.	By employing the use of a lithium ion polymer rechargeable battery for internal power, the system will not require constant maintenance.
The device must be durable enough to withstand the sometimes harsh conditions of an HVAC system.	By mounting the sensors/energy harvesting system against the duct wall as well as having all sensors rated beyond plausible conditions of an HVAC unit, the system will be able to withstand harsh HVAC conditions.
The measured data must be accurate and reliable, even within a wide range of HVAC operating conditions.	The DHT22 is within ±0.5°C and 2-5% humidity. The SDP610-025PA is within 0.004 in. H_2O of actual pressure.
The system must be able to operate on 1.5 ton to 5 ton HVAC systems.	The Savonius wind turbine was designed considering the smallest HVAC duct (18”x24”). Therefore, the retrofitting into existing system as well as future systems is possible.

Table 33. HVAC Monitoring System Engineering Requirements.

Engineering Requirements	Implementation Satisfying Requirement
The device must not impede airflow or reduce the overall effectiveness of the system in any appreciable way.	By implementing a Savonius wind turbine designed specifically to maximize output with minimal input, the system is maximizing effectiveness while minimizing wind interference in the duct. Also, using an internal rechargeable battery, from airflow, allows the device to store the necessary power for operation.
The device must harvest and store only enough energy to operate the system.	
The device must send data about its current functional status, including charge rate and battery life (in percentage)	Implementing a charge controller system allows the microprocessor and charge controller to monitor the overall charge status and charging rate of the battery.
Each component within the system must be able to operate between 0°C and 50°C.	The temperature, air speed, and humidity sensors are well within the allotted temperature frame.
The device must sample data at a reasonable interval (1 sample/5 minutes)	The DHT22 and SDP610-025PA allow the system to keep accurate data measurements. The sampling rate is a sliding window, but 1 sample/5 minutes is well within the power profile of the system given the rate of charge obtainable in the system, battery size, and power requirements of the sensors.
The measured temperature must be within 0.5°C of actual.	
The measured air flow must be within 20 CFM of actual.	
The device must be small enough that it fits in standard HVAC units without significantly affecting airflow. (Current testing unit is 18”x24”)	The Savonius wind turbine was constructed with the large majority of HVAC units and associated ducts in thought. This will allow retrofitting into existing systems as well as assembly into future systems.
The system must be easily mountable.	The device will be anchored in the duct with the durability to withstand the airflow output from the HVAC system.

8.2 Recommendations for Improvement

If the implementation process were to be repeated, there would be a few changes to the system that would be made. The first of these recommendations would be to use a higher quality 3D printing process. The process that was used for this version had difficulties

printing the turbine and, as a result, required multiple attempts for most parts. With better materials and printer, this could have been avoided. This, would have helped resolve the issue with the support on the far wing of the mounting apparatus without needing a redesign. Finally, an improved pressure sensor could be used. The current approach would sometimes give inaccurate readings because of the airflow in the vent being choppy.

If cost savings were desired for this project, there are very few reasonable ways to achieve that. The components were all relatively inexpensive and had no frivolous features that could have been done without. Also, all of the 3D printed material was provided at no cost, which would have been a large expense. Had the printing not done by the university, it likely would have most likely consumed the entirety of the available budget.

9. References

[1] Ott, James H., Anglin, Mark E., “Air flow sensor and detecting method,” U.S. Patent 5 212 983 A, Mar, 18, 1991.

[2] Hammer, Jeffrey M., Lentz, Tracy L., Rotering, Leisha J., Stout, Mark E., “Thermal airflow sensor for an hvac system,” World Patent 2 004 106 864 A1, Dec16, 2002.

[3] Vladimir Cervenka, Lubomir Mraz, and Dan Komosny, “Comprehensive Performance Analysis of Lightweight Mesh and Its Comparison with ZigBee Pro Technology,” Wireless Personal Communications, vol. 78, no. 2, pp. 1527–1538, 2014.

[4] Rabaey, J.M.; Ammer, M.J.; da Silva, J.L., Jr.; Patel, D.; Roundy, S., "PicoRadio supports ad hoc ultra-low power wireless networking," *Computer*, vol.33, no.7, pp.42,48, Jul 2000.

[5] E. Sardini, M. Serpelloni, “Self-powered wireless sensor for air temperature and velocity measurements with energy harvesting capability”, IEEE Transactions on Instrumentation and Measurement, 60(5), (2011), 1838–1844, ISSN: 0018-9456.

[6] Dibin Zhu; Beeby, S.P.; Tudor, M.J.; White, N.M.; Harris, N.R., "Novel Miniature Airflow Energy Harvester for Wireless Sensing Applications in Buildings," *Sensors Journal, IEEE* , vol.13, no.2, pp.691,700, Feb. 2013.

[7] Nelson, Joshua, Purdue University, “Using Piezoelectric Generation to Harvest Energy from Turbulent Air Flow,” Dec, 2007.

[8] U.S. Department of Energy, “Advanced Strategy Guideline: Air Distribution Basics and Duct Design,” Dec, 2011.

[9] Savonius Wind Turbine, https://en.wikipedia.org/wiki/Savonius_wind_turbine, Aug, 2015.

[10] U.S. Department of Energy. April, 2015. “Annual Energy Outlook 2015.” Available: http://www.eia.gov/forecasts/aeo/pdf/0383(2015).pdf [November 7, 2015].

[11] A.A. El-Haroun, N.H. Mahmoud, M.H. Nasef, E. Wahba. (2012, March). “An Experimental Study on Improvement of Savonius Rotor Performance.” *Alexandria Engineering Journal*. [Online]. Available: http://www.researchgate.net/figure/257497943_fig9_(a)-Schematic-diagram-of-Savonius-rotor-connected-with-mechanical-torque-measurement [November 29, 2015].

[12] Magdi Ragheb and Adam M. Ragheb (2011). “Wind Turbines Theory - The Betz Equation and Optimal Rotor Tip Speed Ratio,” *Fundamental and Advanced Topics in Wind Power*, Dr. Rupp Carriveau (Ed.), ISBN: 978- 953-307-508-2, InTech. Available: http://www.intechopen.com/books/fundamental-and-advanced-topicsin-wind-power/wind-turbines-theory-the-betz-equation-and-optimal-rotor-tip-speed-ratio [November 29, 2015].

[13] H. Bahari, N. Goudarzi, W.D. Zhu (2014, May). “An Assessment of the Potential of a Novel Ducted Turbine for Harvesting Wind Power.” *Journal of Intelligent Material Systems and Structures*. [Online]. Available: http://www.researchgate.net/publication/262486649_An_assessment_of_the_potential_of_a_novel_ducted_turbine_for_harvesting_wind_power [November 29, 2015].

[14] CFD Team at IMFD TU Bergakademie Freiberg (2015, April). “Savonius Rotor CFD Simulation with OpenFOAM.” Available: https://www.youtube.com/watch?v=1I4y7rLKdDE [November 30, 2015].

[15] W. Miller. (2014, April). “Predicting Wind Power with Greater Accuracy.” *Science and Technology Review*. [Online]. https://str.llnl.gov/april-2014/miller [November 29, 2015]

[16] https://hvacdealers.com/resources/unit-size-matters/

[17] O. Mojola (1985). “On the Aerodynamic Design of the Savonius Windmill Rotor.” *Journal of Wind Engineering and Industrial Aerodynamics* Available: http://www.sciencedirect.com/science/article/pii/0167610585900054 [November 29, 2015]

[18] J.L. Menet (2004). “A double-step Savonius rotor for local production of electricity.” *Renewable Energy*. Available: http://www.sciencedirect.com/science/article/pii/S0960148104000916 [November 29, 2015].

[19] http://www.veris.com/docs/app/vn48_1009MeasuringAirflow.pdf

[20] https://cdn-shop.adafruit.com/datasheets/XBee+ZB+User+Manual.pdf

Appendix I: Datasheets

Description	Manufacturer Part Number	Datasheet Link
XBee Series 2	XB24-Z7WIT-004	http://www.adafruit.com/datasheets/XBee%20ZB%20User%20Manual.pdf
Atmel ATmega328p	ATMEGA328P-AU	http://www.adafruit.com/datasheets/ATMEGA328P.pdf
Trinket Pro	2010	https://learn.adafruit.com/introducing-pro-trinket
Temperature & Humidity Sensor	DHT22	http://www.adafruit.com/datasheets/DHT22.pdf
Pressure Sensor	SDP610-025PA	http://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/DiffPressure/Sensirion_Differential_Pressure_SDP6x0series_Datasheet_V.1.9.pdf

Appendix II: Pseudo-Code

Microprocessor Pseudo Code (OVERVIEW)

```
---------------------------------------------------------------------------------------------------
/*
setup
for every (sample time)
        send power to sensor
        send power to transmission device

        setup
        Listen for sensor Data
                If data is recieved
                        prepare data for transmission device
                        Send data to transmission device

                        Wait for confirmation from transmission device
                        If Confirmation is recieved
                                Remove power from sensor
                                Remove power from transmission device

if charge controller status is "full"
        sleep
        */
---------------------------------------------------------------------------------------------------
#include <OneWire.h>
#include <DallasTemperature.h>
#include <Wire.h>
#include <SoftwareSerial.h>
#include <avr/sleep.h>
#include <avr/power.h>
#include "TimerOne.h"

#define ONE_WIRE_BUS 5    // defines OneWire pin

        // set pin numbers
const int VIN_PIN = 0;
const int airflowData_PIN = 1;
const int tempData_PIN = 2;
const int batteryData_PIN = 3;
const int airflowSensorPower_PIN = 4;
const int tempSensorPower_PIN = 5;
const int zigbeePower_PIN = 6;
const int zigbeeDataRX_PIN = 7;     //RX
const int zigbeeDataTX_PIN = 8;   //TX
```

```
float sampleTime;


//software serial ports for zigbee, TX = pin 8, RX = pin 7
SoftwareSerial portOne(8, 7);

//Setup a onewire instance to communicate with onewire devices
OneWire oneWire(ONE_WIRE_BUS);

//Pass our oneWire reference to Dallas Temp
DallasTemperature sensor(oneWire);


setpins(){
	//initialize pins to input or output

	//data in
	pinMode(airflowData_PIN, INPUT);    //I2C...
	pinMode(tempData_PIN,INPUT);        //Dallas Single Wire
	pinMode(batteryData_PIN, INPUT);
	pinMode(zigbeeDataIN_PIN,INPUT);

	//power out
	pinMode(airflowSensorPower_PIN, OUTPUT);
	pinMode(tempSensorPower_PIN,OUTPUT);
	pinMode(zigbeePower_PIN, OUTPUT);

	//data out
	pinMode(zigbeeDataOUT_PIN);
}

void sendPower_Devices(){
	//send power to devices
	digitalWrite(airflowSensorPower_PIN,HIGH);
	digitalWrite(tempSensorPower_PIN,HIGH);
	digitalWrite(zigbeePower_PIN,HIGH);
}

void setup(){
	//sets pins
	setpins();

	//starts the hardware serial port
	Serial.begin(9600);  //baud rate
```

```
	//Start up onewire library
	sensors.begin();

	//starts I2C bus
	Wire.begin();

	//starts software serial ports
	portOne.begin(9600);   //baud rate

	//configures timer for sleep
	Timer1.initialize(sampleTime);  // using "timerOne.h"
}                                                                    //used to wake device after sleep

string tempSensor_Listen(){

	sensors.requestTemperatures();  //command to get temperatures
	return(sensors.getTempCByIndex(0));  //index = 0, first IC on the wire
}

char airflowSensor_Listen(){      //rough psuedo code
	//listens for airflow data, returns string of data
	Wire.requestFrom(1, bytes_requested);

	while(Wire.available()){
		char c = Wire.read();
	}

	return(c);
}

bool sendData2zigbee(data){    // return "error", sends data via the bus
	if(portOne.isListening()){
		return true;          //return true to represent error
	}
	else{
		portOne.print(data);
		return false;         //return false to represent no error
	}
}

bool waitforConfirmation(){  //returns int corresponding to received a transmission
	string response;
	while(portOne.available()){
		response = portOne.read();
```

```
                if(response == 'confirmation string')  //checks response from zigbee against
                        return false;           //desired string, returns false for no errors
                if(response != 'confirmation string')
                        return true;                    // returns true for error
        }

}

//organizes data for transmission, returns combined data
string dataModification(tempData, airflowData, batteryData){        string combined;

        combined = tempData + " " + airflowData + " " + batteryData;
        //space to seperate data, could use other symbols as well

        return combined;
}

void removePower_Devices(){
        //remove power to devices
        digitalWrite(airflowSensorPower_PIN,LOW);
        digitalWrite(tempSensorPower_PIN,LOW);
        digitalWrite(zigbeePower_PIN,LOW);
}

int getchargeData(){
        int chargeStatus = analogRead(batteryData_PIN);
        //gets signal from charge controller
        //converts voltage to int 3.3volts/1024 .0032 v/unit
        //status bit can also be read, not as acurate; H or L
         return chargeStatus;
}

void sleep(){
        set_sleep_mode(SLEEP_MODE_PWR_DOWN);  //sets the sleep mode
        sleep_enable();

        Timer1.start(); //starts timer
//wake process will be found in wake function
}

void wake(){
        loop{
        if(Timer1 == sampleTime){         //checks the timer to see when to wake up
                sleep_disable();
                break;              //breaks 'loop'
                }
```

```
        }
}

bool isBatteryCharged(){
        float batteryVoltage = 3.3;    //battery voltage
        float volts2var_unit; // batteryVoltage/1024;  //unit microprocessor can read in
        float fullCharge = batteryVoltage*.98;  //full charge is at 98%, to prevent damage
        float chargeStatus;

        loop{
        //get data from charge controller
        volts2var_unit = getchargeData();

        //converts units from charge Controller to volts2var_unit
        chargeStatus = 1024 / volts2var_unit;

        //if charge is at 98% break loop
         if(chargeStatus >= fullCharge){
                 return true;
                }
        }
}

//=======================Main Loop=================================


char airflowData = null;
int batteryData = null;
string tempData = null;
string data_final = null;
bool transmissionError = false;
bool waitConfirmationError = false;

loop{
        //sets up pins,timer, misc
        setup();

        //sends power to sensors and transmitter
        sendPower_Devices();

        //checks to see if data is recieved
        loop{
                airflowData = airflowSensor_Listen();
                tempData = tempSensor_Listen();

                if(airflowData != null && tempData != null) //makes sure theres data in both
```

```
                    break;                          //breaks out of loop
        }

        //Prepares data for transmission
        data_final = dataModification(tempdata, airflowData, batteryData);

        //Transmit data
        loop{
                //sends data, returns if error happend
                transmissionError = sendData2zigbee(data_final);

                //if error happend, repeat, else break from loop
                if(transmissionError == false)
                        break;
        }

        //waits for confirmation from zigbee
        loop{
                //waits for confirmation, returns error
                waitConfirmationError = waitforConfirmation();

                //checks error, if no error break from loop
                if(waitConfirmationError == false)
                        break;
        }

        //remove power from transmission device and sensors
        removePower_Devices();

        loop{
                //if battery is charged break
                if(isBatteryCharged)
                        break;
        }

        //puts microprocessor to sleep
        sleep();
        //wakes microprocessor if timer is 'up'
        wake();

}
```

Data Center Pseudo Code (OVERVIEW)

```
/*
        PC Software Block Diagram

        Recieve Transmission
                Is Data Complete?
                        Send Confirmation
                        Organize Data
                        Store Data
                        Display Data

                Wait For Transmission

*/
```

```
//MSchwarz developed a Zigbee library that works with C#
using System;
using System.Collections.Generic;
using System.Text;
using MSchwarz.Net.XBee;
using System.Threading;
using System.IO;
using System.Drawing;
using System.Drawing.Imaging;
using MSchwarz.IO;
using System.IO.Ports;

//initializes new a new zigbee device
Xbee xbee_device = new Xbee("COM1", 9600);

//initializes new request
ZigBeeTransmitRequest zRequest = new ZigBeeTransmitRequest('parameters');

//store packet
byte[] recievedPacket = new byte;

//string of data
string dataString = new string;

//bool to see if recieved packet is good
bool packetGood = new bool(false);

//substrings for different data; temp, speed, battery
string[] subStrings= new string[];
```

```
//char delimter to break string up: space
char delimiter = ' ';

//2d array of points for graphing
float[,] points[,] = new float[,];


//converts packets to string
string convertPacket_to_String(recievedPacket[]){

	//sets var to input
	byte[] dataArray = recievedPacket[];

	//string to store data
	string dataStr = new string;


	for(int i = 0; i <= dataArray.size(); i++)
	{
		dataStr[i] = recievedPacket[i].toString();
	}

	return dataString;
}

void storeData(substring){
	 add substring to database;
}


static void Main(string[] args)
{
	//opens communication to zigbee
	xbee.Open();

	//waits to recieve transmission
	while(true)
	{
		//get packet from zigbee
		recievedPacket = zRequest.GetPacket();

		//check to see if packet is good
		//checks parity bit against 'good'
		//if data is bad, ask for retransmission
		if('last bit' != 'good')
```

```
                {
                        xbee_device.SendPacket('resend data');
                        askforRetransmission();
                        packetGood = false;
                }
                else
                {
                        dataString = convertPacket_to_String(recievedPacket[]);
                        packetGood = true;
                }


                //Organize data into substrings
                subStrings = dataString.Split(delimiter);

                //store data,rough psuedo
                foreach(var substring in subStrings)
                        storeData(substring);



                //graph data, rough psuedo
                //get data points from database
                points[,] = getdatafronDataBase();
                //graphs points
                graphPoints(points[,]);


                //calls display of graph
                displayGraphs();


        }
}
```

Appendix III: Matlab Testing Code

```
clc;

WithTurbine = [1.2 3.15 5.1 5.8 6.5 6.5 6.5 5.8 5.1 3.35 1.6 2.8 4 3.95 3.9;
   1.4  2.55  3.7   4.05  4.4   4.275  4.15  4.225  4.3   2.925  1.55  2.35  3.15
        3.025 2.9;
   1.6  1.95  2.3   2.3   2.3   2.05   1.8   2.65   3.5   2.5    1.5   1.9   2.3
        2.1   1.9;
1.15    1.425 1.7   1.95  2.2   2.425  2.65  2.475  2.3   1.875  1.45  1.675 1.9
        1.975 2.05;
0.7     0.9   1.1   1.6   2.1   2.8    3.5   2.3    1.1   1.25   1.4   1.45  1.5
        1.85  2.2;
1.3     1.675 2.05  3.025 4     5.175  6.35  4.675  3     2.65   2.3   2.225 2.15
        2.25  2.35;
1.9     2.45  3     4.45  5.9   7.55   9.2   7.05   4.9   4.05   3.2   3     2.8
        2.65  2.5;
2.95    3.375 3.8   5.2   6.6   8.125  9.65  8.45   7.25  6.375  5.5   5.025 4.55
        4.125 3.7;
4       4.3   4.6   5.95  7.3   8.7    10.1  9.85   9.6   8.7    7.8   7.05  6.3
        5.6   4.9]

figure(1)
contourf(WithTurbine,40,'LineColor', 'none');
caxis([0 11.1]);
h = colorbar;
set( h, 'YDir');
title('With Turbine');

WithoutTurbine = [1.4       3.5   5.6   7.1   8.6   9.8   11    10.8  10.6  8.2
        5.8   5.9   6     4.15  2.3;
1.35    3.625 5.9   7.1   8.3   9.675  11.05 10.2  9.35  7.6   5.85  5.275 4.7
        3.275 1.85;
1.3     3.75  6.2   7.1   8     9.55   11.1  9.6   8.1   7     5.9   4.65  3.4
        2.4   1.4;
1.15    3.375 5.6   6.7   7.8   9.075  10.35 8.15  5.95  5.3   4.65  3.775 2.9
        2.175 1.45;
1       3     5     6.3   7.6   8.6    9.6   6.7   3.8   3.6   3.4   2.9   2.4
        1.95  1.5;
1.1     2.325 3.55  5.25  6.95  8.35   9.75  6.325 2.9   2.675 2.45  2.2   1.95
        1.875 1.8;
1.2     1.65  2.1   4.2   6.3   8.1    9.9   5.95  2     1.75  1.5   1.5   1.5
        1.8   2.1;
1.25    1.975 2.7   5.2   7.7   8.475  9.25  5.65  2.05  1.675 1.3   1.4   1.5
        1.9   2.3;
```

```
1.3     2.3     3.3     6.2     9.1     8.85    8.6     5.35    2.1     1.6     1.1     1.3     1.5
        2       2.5]

figure(2)
contourf(WithoutTurbine,40,'LineColor', 'none');
caxis([0 11.1]);
h = colorbar;
set( h, 'YDir');
title('Without Turbine');

Difference = WithTurbine-WithoutTurbine;
figure(3)
contourf(Difference,40,'LineColor', 'none');
caxis([-9.3 7.5]);
h = colorbar;
colormap(hot);
set( h, 'YDir');
title('Difference');
```

Appendix IV: Sensor Testing Code

Temperature/Humidity Sensor Code

```
#include "DHT.h"

#define DHTPIN 3     // what digital pin we're connected to
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
}

void loop() {
delay(2000);

  float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  float f = dht.readTemperature(true);

  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t) || isnan(f))
  {
    //digitalWrite(13, HIGH);
    Serial.println("Failed to read");
    return;
  }
  else
  {
   // Serial.println("Reading...");
   /* Serial.print("Data Point: ");
    Serial.print(counter);
    Serial.println("");
    */counter++;
    //digitalWrite(13, LOW);
  }
  // Compute heat index in Fahrenheit (the default)
 float hif = dht.computeHeatIndex(f, h);
  // Compute heat index in Celsius (isFahreheit = false)
 float hic = dht.computeHeatIndex(t, h, false);

  Serial.print("Humidity: ");
  Serial.print(h);
```

```
  Serial.print("%");
  Serial.println("");

  //Serial.print(" %\t");
  Serial.print("Temperature: ");
  //Serial.print(t);
  //Serial.print(" *C ");
  Serial.print(f);
  Serial.print(" *F");
  Serial.println("");
  ///Serial.print("Heat index: ");
  //Serial.print(hic);
 // Serial.print(" *C ");
 // Serial.print(hif);
 // Serial.print(" *F");
  Serial.println("----------------------");

}
```

Pressure Sensor Code

```
#include <Wire.h>


void setup()
{
  Wire.begin();

  Serial.begin(9600);
  while (!Serial);             // Leonardo: wait for serial monitor
  Serial.println("\nI2C Scanner");
}


void loop()
{
  byte error, address;
  int nDevices;

  Serial.println("Scanning...");

  nDevices = 0;
  for(address = 1; address < 127; address++ )
  {
    // The I2C_scanner uses the return value of
    // the Write.endTransmisstion to see if
```

```
    // a device did acknowledge to the address.
    Wire.beginTransmission(address);
    error = Wire.endTransmission();

    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.print(address,HEX);
      Serial.println("  !");

      nDevices++;
    }
    else if (error==4)
    {
      Serial.print("Unknow error at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.println(address,HEX);
    }
  }
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");

  delay(5000);           // wait 5 seconds for next scan
}
```