Jason Gass
Adam Fobbe
Saksham Ghimire
Brett Tiffany
Nick Bachman

University of Wisconsin - Stout
Engineering and Technology Department
Computer Engineering Program
University of Wisconsin - Stout

TABLE OF CONTENTS

ABSTRACT

“To design an Air Quality Monitor device that is wearable and can communicate over Bluetooth with an Android device to display and record various air quality sensor readings.”

ACKNOWLEDGEMENTS

We would like to mention those who have helped or assisted us in our educational journey and specifically with this senior project. We would like to give a special thanks to professors Dr. Cheng Liu and Dr. Andy Peng from UW-Stout who have helped our team out a great deal in our engineering path through instruction, examples and taking the time to encourage the students.

1. INTRODUCTION

The medical device industry in today’s world plays a significant role in improving lives of patients. From revolutionary pacemakers to state of the art diagnostic devices, the healthcare industry has been adopting new technologies to treat patients and help them take better care of themselves. The National Science Foundation (NSF) has been an active organization to help researchers bring new technologies to the market and help healthcare organizations recommend these products to patients.

Most of today’s medical devices have been credited for being proactive in the development of health care facilities both in terms of planning and building of new treatment centers and upgrading and improvement of existing facilities. This also depends on the type of health problem being treated since some medical devices are dedicated to help improve specific health problems. According to World Health Organization (WHO), “Medical devices are indispensable for effective prevention, diagnosis, treatment and rehabilitation of illness and disease”.

Although, the medical device industry has been able to cover most of the common worldwide health problems, there are still a few health problems that require immediate attention of the medical device industry to help patients become proactive and be able to monitor their health in real time. The NSF has been actively involved

in helping researchers develop innovative ways of helping Asthma patients. The patients diagnosed with asthma have to be extremely careful about the environment conditions they are trying to adapt in. Factors such as temperature, humidity, NO_2 and ozone present in the atmosphere can trigger asthma related attacks. The patient has to be aware of the existing atmospheric conditions and avoid any conditions that might pose a threat towards their health condition. By far, no medical device companies have come up with a concrete solution to help the asthma patients monitor the environment in real time.

By applying concepts from computer and electrical engineering, we as engineers are professionally obligated to help solve this problem for asthma patients. The need for a medical device that can constantly monitor atmospheric conditions and instantly notify asthma patients must be fulfilled to resolve the problem.

2. PROBLEM STATEMENT AND SPECIFICATIONS

We have a need to research and develop a medical device to help asthma patients constantly monitor atmospheric conditions and get real time updates about the condition.

The device must be able to sense and consider all relevant atmospheric factors and process them with high precision and accuracy. The device must be portable and should communicate wirelessly with the patient's smart phone to synchronize information from the device. The health monitor device must also be able to store patients data securely which can later be retrieved by an authorized personnel. The device must follow FDA regulations in the United States and must be able to pass health regulations in other countries internationally.

The specifications for this project are as follows:

1. Communicate to an android device via Bluetooth
2. Get atmospheric condition updates using sensors such as GPS, temperature, humidity, NO_2 and ozone.
3. Design an ergonomic enclosure
4. Supply power using a rechargeable battery
5. Interface a storage unit for sensor data
6. Warn users of high exposure levels
7. Design a single PCB

2.1 General Design Approach

From initial research, it was clear that there are a few devices in the market that are commercially available and are able to read various atmospheric conditions. They were inexpensive and simple to use. Considering the cost and the simplicity of these

existing products, they did have some drawbacks. These products are designed to measure only a specific factor that contributes towards air quality. To measure all factors that come into play when air quality has to be tested, all the devices would have to be separately and have they work together. This would add up on the cost of the device and would not be portable at all. Since, these devices are built for specific purposes, it would not be feasible to have them all work together without avoiding compatibility issues and other technical difficulties.

To serve the purpose of this project to the fullest, a device was needed that met all the specifications and is specifically customized to help asthma patients. Since, there were no devices out on the market that are specifically designed to help asthma patients and met the project specifications, the group's decision was to build one and fully customize it to help asthma patients.

The temperature and humidity sensors were very straight forward and the group had no issues setting them up. Meanwhile, android code was developed that would help any android device communicate with the device. The GPS and dust sensors were tricky and required more troubleshooting. A voltage divider was used to control the voltage driving into the NO2 and Ozone sensors. The Bluetooth 2.0 dongle we used worked seamlessly to connect the board with an Android smartphone.

2.2 Specifications

The success of the project hugely depends on the preliminary requirements stated in the project definition. Since, this is a medical device, extra layers of safety and precaution have to be taken into account to prevent any accidental damage. The patient will be carrying the device in a very close proximity which increases the safety factors related to the device. The technical specifications are as follows:

1. The device must be able to communicate to an android device via Bluetooth
2. The device must be able to measure air quality using sensors such as GPS, temperature, humidity, NO_2 and ozone.
3. The device must be enclosed inside a safe and ergonomic enclosure
4. The device must be able to run using a rechargeable battery
5. The device must be able to store patients data using latest flash memory storage technology
6. The device must regularly and seamlessly work with the software on the patient's phone and warn them of high exposure levels.
7. The device must have a fallback plan such as a reset option in cases of malfunction

8. The device must be lightweight to carry and must be comfortable to carry around for longer periods of time
9. The device must use low power to keep the battery from draining out quickly
10. The device must be reasonably affordable

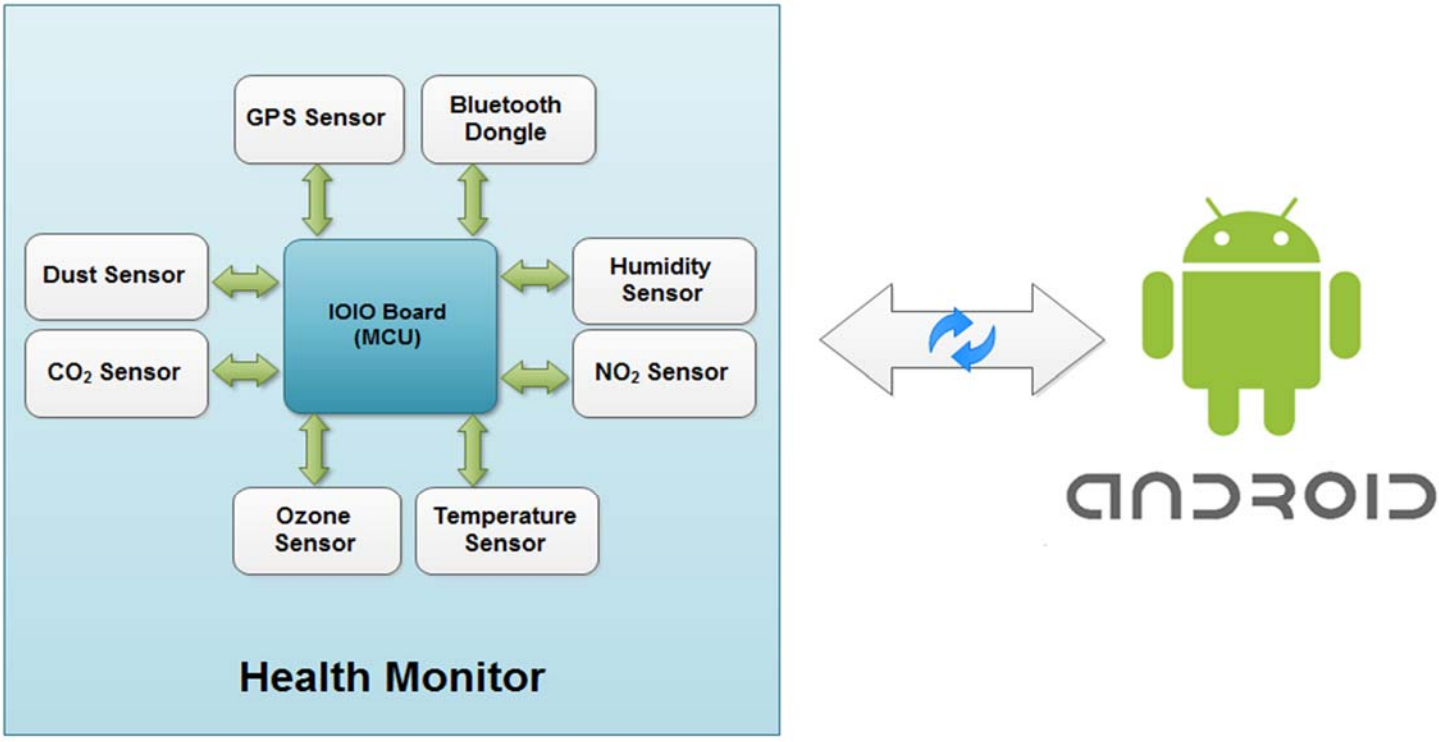


Fig: Block diagram showing the interaction of sensors with the IOIO board

The implementation of wireless technology in medical devices has many benefits such as patient mobility and to avoid being on the bed for longer periods of time. Also, wireless medical devices can be remotely programmed and physicians are able to access patient data from anywhere. By allowing remote monitoring physicians are able to help more number of patients in real time. Also patient's benefit from wireless medical devices because of its portability. As engineers, who have taken the task of building a medical device to help asthma patients we need to be extremely careful about what kind of wireless technology we are implementing, the kind of risks we are taking in case of any accidental damage, data security and most importantly, the Electromagnetic Compatibility (EMC).

In cases of electronic medical devices, the Federal Communications Commission (FCC) and the Food and Drug Administration (FDA) work together to insure the quality of electronic devices. The FCC also oversees the use of Radio Frequency (RF) within which various RF wireless technologies operate. For this project, the only wireless

technology we have used is the Bluetooth chip to connect the health monitor to an android phone. According to the FCC, any Bluetooth enabled device must be a short range broadcast equipment operating between 2.4 to 2.485 GHz, using a spread spectrum, frequency hopping, full-duplex signal at a nominal rate of 1600 hops/sec. The 2.4 GHz ISM band is available and unlicensed in most countries. The FDA along with the FCC also requires that the authorized personnel for prescription of medical devices using wireless radio frequency are obligated to notify/warn customers about incorporated radio frequency. The FCC also has a list of exceptional conditions which allows certain unintentional radiators. They are codified in 47 CFR 15.103 and are exactly as follows:

- *Digital devices oscillating below 1.705 MHz that do not connect to the power grid, even indirectly,*
- *To be exempt, devices also cannot connect for the purpose of recharging batteries.*
- *Digital devices that use less than 6 billionths of a watt (6 nW) of electrical power*
- *Devices only used in vehicles*
- *Specialized medical, electrical utility or commercial test & measurement devices.*
- *Appliances (white goods), or devices used exclusively in appliances.*
- *Non-digital simple passive devices*

2.3 Project Schedule

To keep the project on schedule a Gantt chart was used. The chart divided all tasks to members of the group and gave deadlines. The first part of the schedule designated a lot of time for research. The group worked on researching new additions. Adam and Saksham focused on sensors, memory, and application design. Brett and Nick then researched board design and companies to manufacture boards. We then designated more specific different tasks to each team member. Once research was finished Adam and Saksham began implementing all of ideas and designing the application. Brett and Nick decided to print off boards with UW-Stout's FabLab machines so this process then began.

The attached Gantt chart will give more specific details pertaining to the project schedule. There wasn't much changed on the Gantt chart throughout the semester it stayed very much the same.

3. DESIGN CHOICES AND PERFORMANCE CRITERIA

There were many design choices involved in this project. One consideration was which IO board to choose. There are many boards on the market but some considerations that were taken were size, cost, communication ports, supply voltage and input/output connections.

The first board we looked at was the Arduino Mega ADK board. This board could do everything that we wanted it to and had many input/output pins to use. The main downside to this board though was that it was just too large for us to use in an enclosure. The second board that we looked at was the IOIO Mint board. This board was small and came in a very nice enclosure. It had fewer pins but would still work for our needs. The third and final board we decided to use was the IOIO for Android board. This board had a very simple design and was easy to get started with. It is also the smallest in size out of the boards we had previously looked at so that played a large part in our choice.

One more design consideration was what type of battery to use because this device was going to be portable. The IO board that was chosen needed 5VDC to operate so a battery greater than 5 volts was needed. A “Portable Power Bank External 2600mAh Mobile USB” battery was chosen. This battery offers the 5 volts we need and offered enough power to run the device for long periods of time without charging. The battery also include a built in micro-USB charging port.

Another consideration for the project was what type of sensors to order. We used several different sensors in this device including a temperature sensor, a humidity sensor, an Ozone (O_3), a Nitrogen dioxide (NO_2) sensor, and a dust/smoke sensor. The TMP36 temperature sensor was chosen because of its low voltage operation, accuracy and low heat output. The TMP36 operates between 2.7V to 5.5V and is accurate to within 2^{o}C. Another benefit of this sensor was that it has a very low self-heating temperature, only 0.1^{o}C in still air. The HIH-4030 humidity sensor was chosen because of its low power consumption, fast response time and accuracy. The HIH-4030 operates from 4V to 5.8V and is accurate within 3.5%. The response time for the humidity sensor is 5 seconds. The MiCS-2610 Ozone sensor was chosen because it has a wide sensing range and high sensitivity. The MiCS-2610 sensor detects particles ranging from 10 to 1000 parts per billion. The sensitivity factor of the MiCS-2610 is 1.5 to 4. The MiCS-2710 Nitrogen dioxide sensor was chosen because it has high sensitivity and a wide detection range. The sensitivity factor of the MiCS-2710 is 6 to 100 and the detection range is between 0.05 and 5 parts per million.

4. DETAILS OF THE DESIGN

4.1 Final Design Details

The Android IO Health Monitor was picked up from a previous group which already had some devices working. The name was later changed to the current “Air Quality Monitor” title with the scope of the project changing as well. This also allowed us to make adjustments as we needed with our own device.

The temperature and humidity sensors were operational and were transmitting signals to an android device via Bluetooth. The GPS module was somewhat working but needed more programming to be operational. A lot of research was done to choose the electronic components and an ergonomic enclosure which was described in Section 3 of this report. Once the sensors were received they were wired and tested. Wiring of the sensors took some time because of the complexity of the wiring circuits. The data sheets for the sensors were referenced many times during the installation process (see Appendix 7.3). Also the output for the IO board was 5 VDC and the ozone sensor required a heating voltage of 2.35 VDC so calculations were done to produce the correct voltage for the sensor. The same was also true for the NO2 sensor, so more calculations were done to get to the correct voltages (see Appendix 7.6). Resistors were installed to reduce the output voltage from the board to the sensors. Once the sensors were installed and working the java programming was started.

There was extensive java programming done to make the sensors communicate correctly to the android device. Research was done to determine what the best way was to work with the sensors. There was not much that was found but through a trial and error processes we were able to get them working correct and efficiently.

The GPS sensor was later taken off to create more space and the alternative option of using the Android device's GPS location was used.

A printed circuit board was also designed. The initial plan was to make a single PCB with the IOIO board and all sensors integrated together. The schematics and design was already with all the parts ordered. The FabLab machine was then found to be unable to make small enough traces for the microcontroller we planned to use. The process took a while because the machine took a long time to cut out the boards. Once we found that it will not work to use that machine it was already too late and the funds were all gone. This caused us to have to design a breakout board to be attached to the IOIO board.

An enclosure was built to safely hold all of the electronic components. Initially, a lot of researches have been done in order to know how to build the enclosure and how it is going to be used by the patient. A lot of estimated sketches were done in the beginning so as to know how the enclosure will be in the future. It was difficult to determine the right measurements because the project was in its infancy. However, after the group members found some of the needed parts on the internet, the correct measurements of the sketch slowly started to become clear. Then, a simple design was made using solidworks. The members of the squad decided to use solidworks instead of AutoCad because of its 3D rendering capability while AutoCad is mostly good for 2D drafting work. Moreover, solidworks was the first choice because one of the group members has used solidworks before.

“How to build the enclosure? What are the right materials?”, were the big questions in the beginning, but when the group searched about the proper materials that can meet our standards and read more about some enclosures that have been built in the past.

The group preferred to use the ABS (Acrylonitrile Butadiene Styrene) material because of its functionality. We went through the data sheet of ABS material and found how efficient is it and how it meets our standards (see appendix 7.3). The material is made of acrylonitrile, butadiene and styrene. It has a high mechanical strength and can handle high temperature. This material is being used in the plastics lab at UW-Stout. With help of the lab assistant we printed our first prototype using the 3D printing machine which is called; Fused Deposition Modeling (FDM) as shown below:

The FDM machine is one of the most popular 3D printers available. Basically, you can just send the solidworks file to the machine and it will directly receive the print design. Then by inserting the ABS material inside the machine it automatically starts building the required 3D shape in a specific time, depending on the size of the prototype (Hiemenz, 2012).

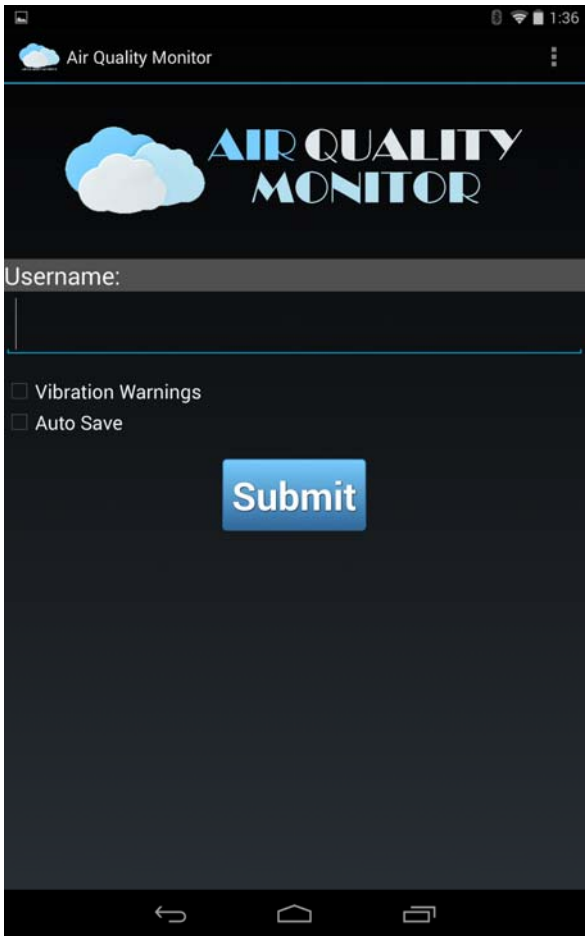
The first prototype was somehow big in size compared to the parts that were going to fit in. A lot of ideas were explored to modify the enclosure such as; making the columns which holds the PCB board and IOIO board taller, adding a slot for the battery charger, replacing the big slot of the SD card with a smaller one, making the slot of the ventilation smaller, removing the hole which was for the wires, making a sliding cover and making the walls thinner.

When we printed the second draft; it turned out nice and great except for the sliding cover which was not going all the way through the end. Once again a lot of research was done to figure out the right way to build the cover. Finally, after the final

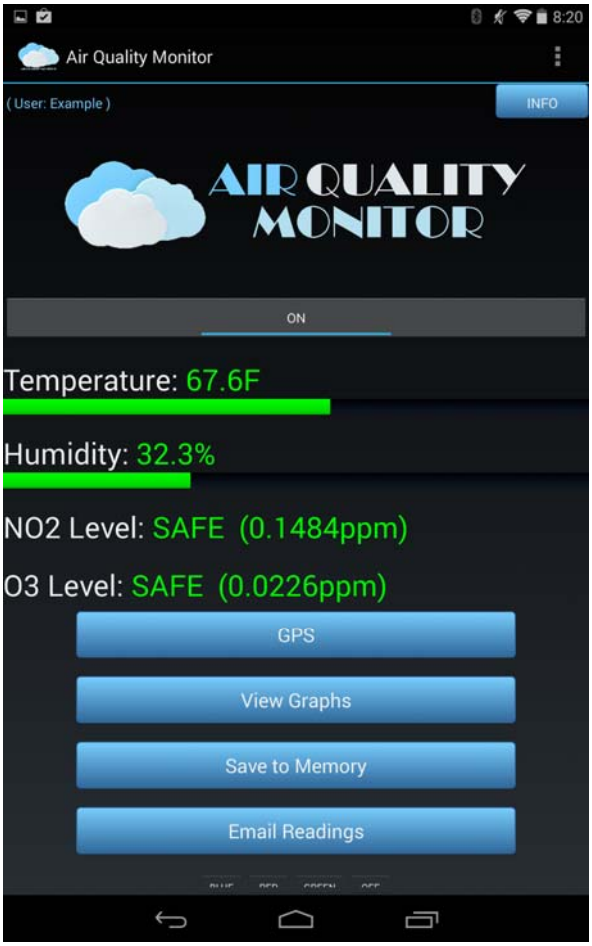
touches and editing the new design was made as shown below:

The fourth design ended up to be the enclosure we used. This design was then printed in black and had “Air Quality Monitor” engraved. This enclosure includes a snap-on cover, Micro-USB charging port, and an ON-OFF switch.

The application design was the largest part in this project. In the end it included seven different screens. When started up it initially asks the user to enter in their name and select if they would like to turn on Vibration Warnings so when a reading reaches an unsafe level it will warn the user by vibrating. It also gives the option to save automatically.

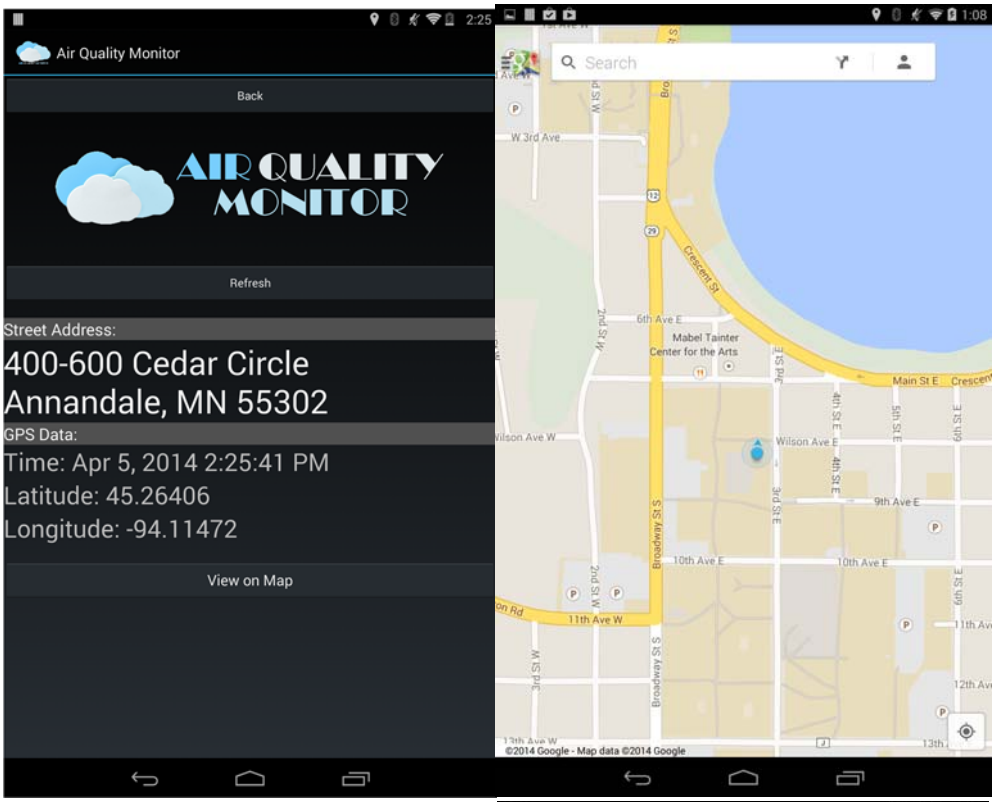


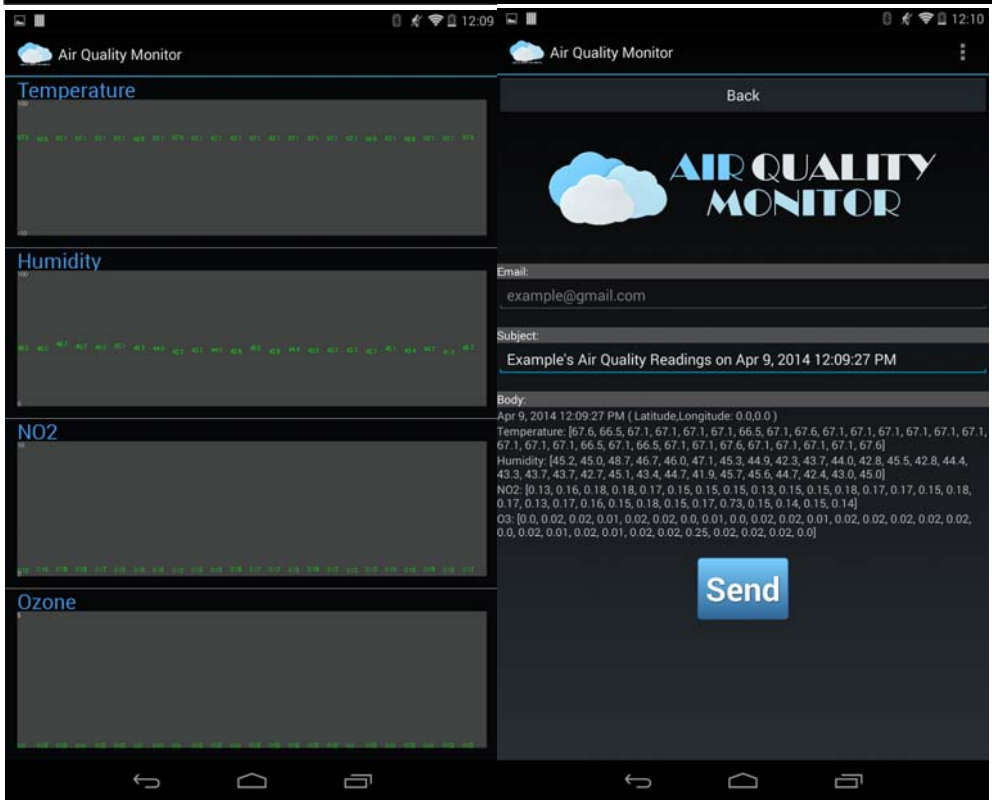
Then once the user clicks submit it brings you to the main reading screen where all the important information is shown. This screen displays the readings in real-time with color indicators. From this screen you can access all the other screens.



“GPS” shows your location along with the option to “View on Map”.
“View Graphs” shows your data graphed.
“Save to Memory” saves your data onto SD storage within the device.
“Email Readings” brings up email to send out your data.

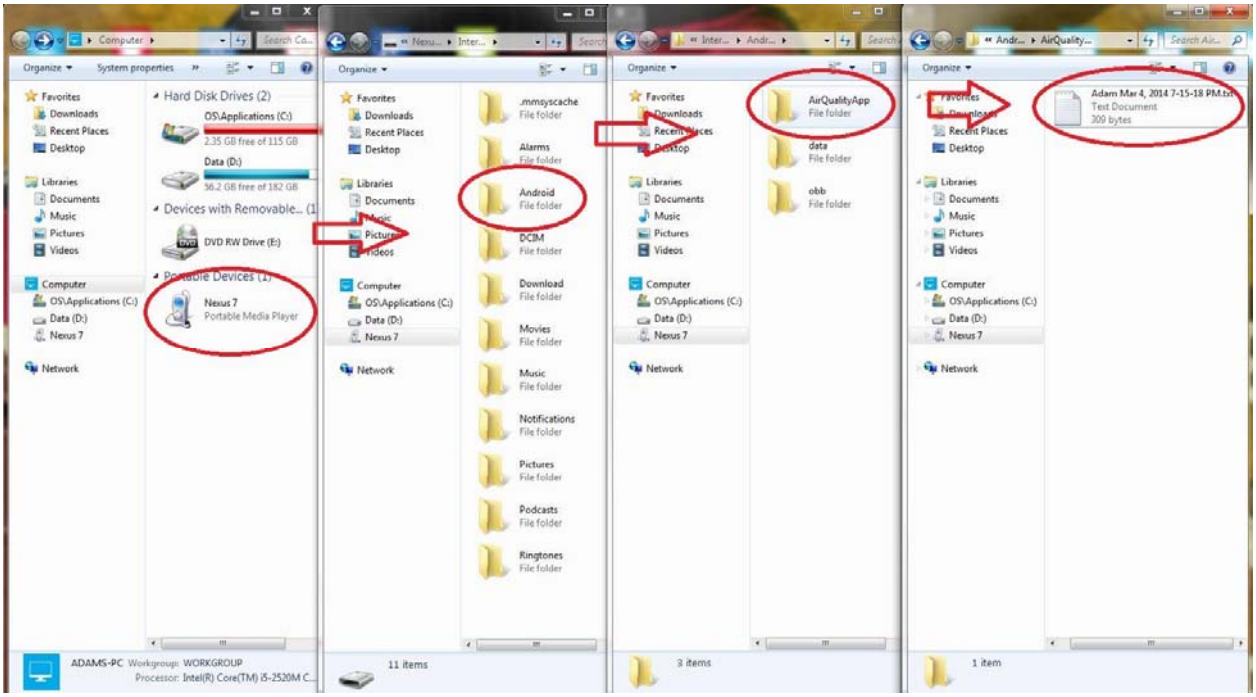
These screens are shown below on next page.



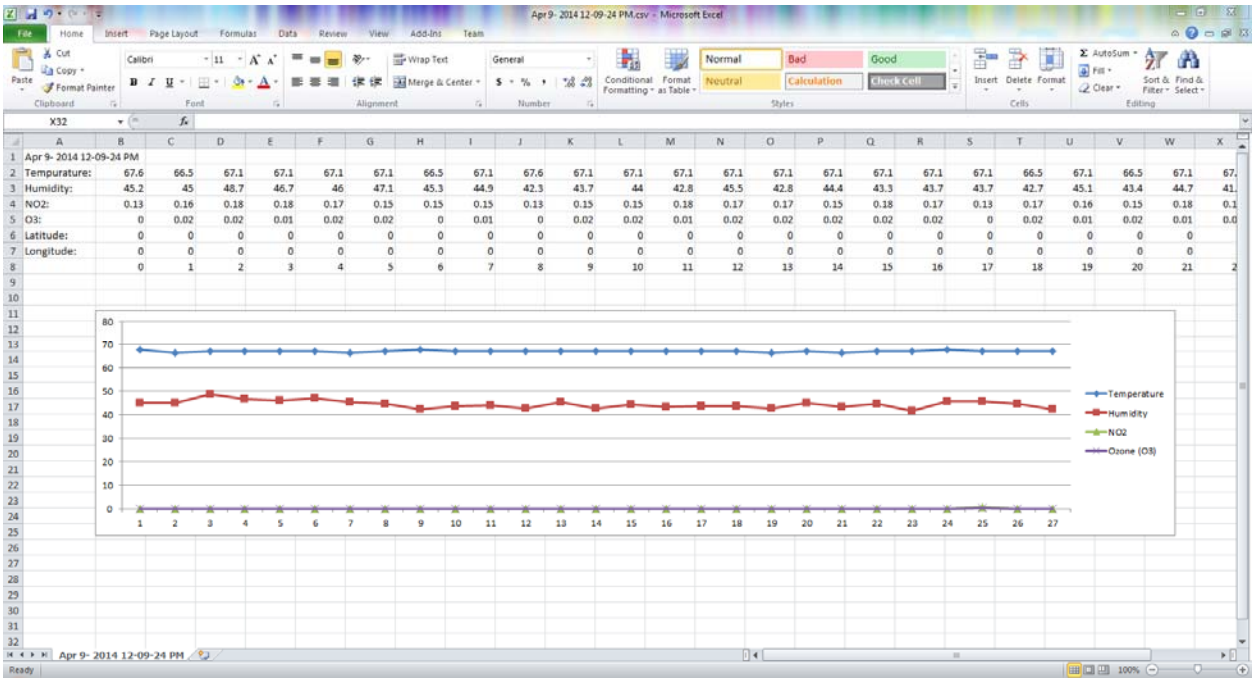


Saved data can then also be accessed and graphed on a computer by connecting it

with a USB cable. To access this data first plug device in and then follow these steps:

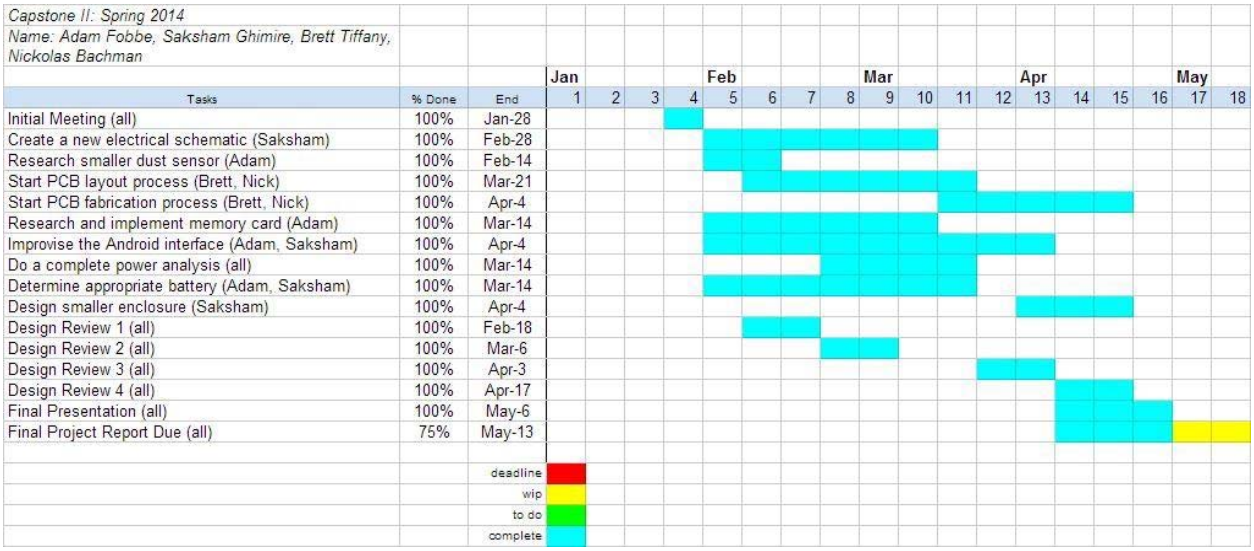


The .csv file can then be opened in Excel to be graphed similar to one below.



4.2 Design Tasks for each Member

Adam and Saksham focused on the software side while Brett and Nick focused on the hardware side. Adam and Saksham wrote code and designed the application. Along with this they designed the enclosure, as well as researching batteries and deciding which to use. Brett and Nick researched and designed the PCB boards to be used. More information on specific tasks in listed in the attached Gantt chart below.



Capstone II: Spring 2014			Jan				Feb				Mar				Apr					May	
Name: Adam Fobbe, Saksham Ghimire, Brett Tiffany, Nickolas Bachman																					
Tasks	% Done	End	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
Initial Meeting (all)	100%	Jan-28																			
Create a new electrical schematic (Saksham)	100%	Feb-28																			
Research smaller dust sensor (Adam)	100%	Feb-14																			
Start PCB layout process (Brett, Nick)	100%	Mar-21																			
Start PCB fabrication process (Brett, Nick)	100%	Apr-4																			
Research and implement memory card (Adam)	100%	Mar-14																			
Improvise the Android interface (Adam, Saksham)	100%	Apr-4																			
Do a complete power analysis (all)	100%	Mar-14																			
Determine appropriate battery (Adam, Saksham)	100%	Mar-14																			
Design smaller enclosure (Saksham)	100%	Apr-4																			
Design Review 1 (all)	100%	Feb-18																			
Design Review 2 (all)	100%	Mar-6																			
Design Review 3 (all)	100%	Apr-3																			
Design Review 4 (all)	100%	Apr-17																			
Final Presentation (all)	100%	May-6																			
Final Project Report Due (all)	75%	May-13																			
		deadline																			
		wip																			
		to do																			
		complete																			

4.3 Test Results and Conclusions

Throughout the semester we have had to constantly test different elements of our project. We began by testing all of our electrical components with a multimeter to ensure they were working properly before we constructed our circuits.

For both the temperature and humidity sensors we tested them before and after we installed them to make sure they functioned properly. Before we installed them we used a multimeter to confirm that they were functioning by making sure there was a voltage reading across the sensor. After the sensors were installed we plugged the output into an equation to determine the temperature and humidity readings and compared these to the actual temperature and humidity readings to ensure they were correct.

The coding involved initializing a pin for each to read the voltage. The temperature sensor reads in Celsius so we had to convert that to Fahrenheit. The humidity sensor uses a formula along with the temperature. So when we got that all written we tested to make sure it was working correctly.

The Bluetooth was working intermittently in the beginning and we were not exactly sure why one Bluetooth unit would work and the other would not. After some testing, we separated out the working Bluetooth modules from the faulty ones. We

never encountered any problems after using the ones that were separated from the faulty units.

We tested our enclosure for a few key requirements, one thing that needed to be tested was the size of the container, and we tested this by installing all of the components to make sure they all fit. We also had to make sure the tolerances for the lid and container were within limits, this was tested simply by installing and removing the lid several times in doing this we also could determine if the slots where the lid fit were durable enough to withstand removing and reinstalling the lid.

As for the gas sensors we developed tests for both gas sensors, but only ended up being able to test the Nitrogen dioxide sensor. In order to test the gasses we had to build an airtight chamber to control the flow of gasses over the sensors. We obtained an airtight glad container, drilled two holes on opposite ends of the container and ran ¼" tubing out of both holes to allow for air to be connected and flow continuously over the sensors. In order to test the Nitrogen dioxide sensors we had to create Nitrogen dioxide by combining copper and nitric acid. We placed our device into the airtight chamber and connected a flask containing a small amount of copper shavings to a hose and an airline in order to allow the gasses to disperse faster. We used a syringe to combine the Nitric acid with the copper shavings by adding one drop at a time in order to produce small enough quantities of gas so as not to saturate the sensor. We have only obtained qualitative data from this test so far, but we believe in the future that we could determine the exact amount of gas the sensor is exposed to and make sure the sensors are calibrated correctly.

The test we created for the Ozone sensor was very similar to the test for the Nitrogen dioxide. We will use the same airtight chamber and other equipment to perform the quantitative testing. In order to produce the Ozone gas we will use oxygen and an electric spark in a flask and use the same technique of running air through the hoses to move the gas across the sensor. As with the Nitrogen dioxide this test will only be qualitative not quantitative, but will still be a valuable test in determining that the sensor does indeed work. The testing code involved reading the voltage of two pins and taking the difference. This gave us the millivolt equivalent of the sensors reading in PPM (parts per million).

5. CONCLUSIONS AND SUMMARY

In conclusion we ended up producing a functional Air Quality Monitor for patients. Our recommendation for the product is to make it more compact so it can be easily carried. We think this product will be very useful and drastically improve asthma patients' quality of life by allowing them to continually track their environmental factors and limit their exposure to hazardous elements. The future for this device is very good and we believe it will positively impact peoples' lives and prove to be a useful and successful product in the medical device industry.

6. REFERENCES (OR BIBLIOGRAPHY)

- World Health Organization. (2010). Medical devices. Retrieved 12/15/2013, from http://www.who.int/mediacentre/factsheets/fs346/en/index.html

- Federal Communications Commission. (2010). Frequently asked questions. Retrieved 12/06/2013, from http://www.bureauveritas.com/wps/wcm/connect/31b53f004edb713e8d5fcd600bbc220b/FCC_Frequently_Asked_Questions_Aug10.pdf?MOD=AJPERES

- Oocities Organization. (2009). Why use SolidWorks. Retrieved 12/08/2013, from http://www.oocities.org/wpsmoke/solidworkswhy.html

- Hiemenz, Joe. (2012). Excerpt 2: 3D printing with FDM Technology: How it Works. Retrieved 12/03/2013, from http://blog.stratasys.com/2012/09/08/excerpt-2-3d-printing-with-fdm-technology-how-it-works/

- OSHA (2013). Limits for Air Contaminants. retrieved Oct. 2013. from https://www.osha.gov/pls/oshaweb/owadisp.show_document?p_table=standards&p_id=9992

7. APPENDICES:

Appendix 7.1 Bill of Materials (BOM)

Single Unit Cost

Find	Description	Part Number	Supplier	Cost	
1	Bluetooth Dongle	BT-V2-M	Sanoxy	$ 2.20	
2	IOIO Board	DEV_10748	Sparkfun	$ 39.95	
3	Temperature Sensor	TMP36	AnalogDevices	$ 1.50	
4	Humidity Sensor	HIH-4030/31	Honeywell	$ 8.52	
5	NO2 Sensor	MiCS-2710	CDI	$ 6.18	
6	O3 Sensor	MiCS-2610	CDI	$ 6.18	
7	Enclosure		UW-Stout	$ 20.00	*
8	Custom PCB		UW-Stout	$ 20.00	*
			Total Cost =	$ 104.53	

Appendix 7.2 Data Sheets and Vendor Specifications

PART	DATASHEET LINK
Temperature Sensor	http://www.analog.com/static/imported-files/data_sheets/TMP35_36_37.pdf
Humidity Sensor	https://www.sparkfun.com/datasheets/Sensors/Weather/SEN-09569-HIH-4030-datasheet.pdf
Ozone Sensor	http://airqualityegg.wikispaces.com/file/view/mics-2610+-+O3.pdf
NO_2 Sensor	http://www.cdiweb.com/datasheets/e2v/mics-2710.pdf

e2v

MiCS-2710
NO_2 Sensor

This datasheet describes the use of the MiCS-2710. The package and the mode of operation illustrated in this document target the detection of nitrogen dioxide (NO_2).

FEATURES

- Low heater current
- Wide detection range
- High sensitivity
- Fast thermal response
- Miniature dimensions
- High resistance to shocks and vibrations

IMPORTANT PRECAUTIONS

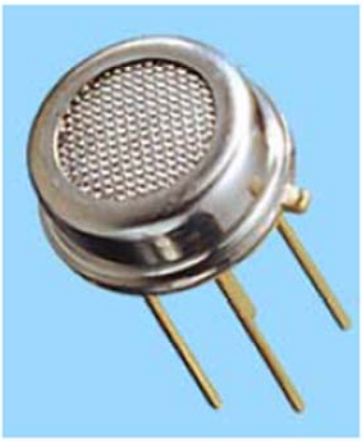
Read the following instructions carefully before using the MiCS-2710 sensor described in this document to avoid erroneous readings and to prevent the device from permanent damage.

- The sensor must not be wave soldered without protection, or exposed to high concentrations of organic solvents, ammonia, or silicone vapours, to avoid poisoning the sensitive layer.
- Heating powers above the maximum rating of 120 mW can destroy the sensor due to overheating.
- This sensor is to be placed in a filtered package that protects it against any water or dust projection.
- For any additional questions, email enquiries@e2v.com or telephone +44 (0)1245 493493.

OPERATING MODE

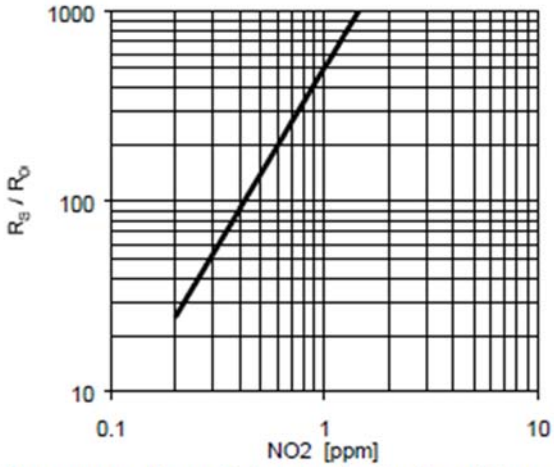
The recommended mode of operation is a constant power mode. A heater power of P_H = 43 mW is applied. This causes the temperature of the sensing resistor (R_S) to reach about 220 °C.

Detection of the pollution gases is achieved by measuring the sensing resistor R_S during operation.



SENSOR RESPONSE

The sensor response to NO_2 in air is represented in Fig. 1.



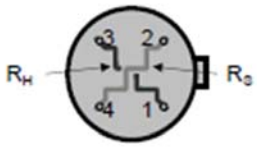
The sensor resistance R_S is normalised to the resistance under air (R_0).

Fig. 1: R_S/R_0 as a function of gas concentration at <5% RH and 25 °C.

MEASUREMENT CIRCUIT

Fig. 2 shows the pin connections of the MiCS-2710 gas sensor. A simple circuit to measure the pollution level is proposed in Fig. 3. The heating voltage V_H is applied to pins 3 and 1. A load resistor R_L is connected in series with R_S to convert the resistance R_S to a voltage V_S between pins 2 and 4. R_S can then be calculated by the following expression:

$$R_S = R_L / (V_{CC} - V_S) \times V_S$$



Pin	Connection
1	Heater ground
2	Sensor pin
3	Heater power
4	Sensor pin

Fig. 2: Equivalent circuit of MiCS-2710 (top view)

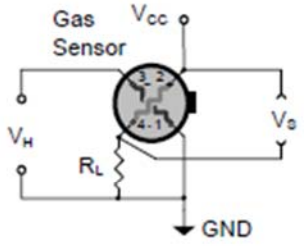


Fig. 3: Measurement circuit for pollution gas detection

ELECTRICAL SPECIFICATIONS

Maximum Ratings

Rating	Symbol	Value/ Range	Unit
Maximum sensor supply voltage	V_{CC}	2.5	V
Maximum heater power dissipation	P_H	50	mW
Maximum sensor power dissipation	P_S	1	mW
Relative humidity range	R_H	5 – 95	%RH
Ambient operating temperature	T_{amb}	-30 – 85	°C
Storage temperature range	T_{sto}	-40 – 120	°C
Storage humidity range	RH_{sto}	5 – 95	%RH

Operating Conditions

Parameter	Symbol	Typ	Min	Max	Unit
Heating power	P_H	43	30	50	mW
Heating voltage	V_H	1.7	-	-	V
Heating current	I_H	26	-	-	mA
Heating resistance	R_H	66	59	73	Ω

Sensitivity Characteristics

Characteristic	Symbol	Typ	Min	Max	Unit
NO_2 detection range	FS		0.05	5	ppm
Sensing resistance in air (see note 1)	R_0	2.2	0.8	8	kΩ
Sensitivity factor (see note 2)	S_R	55	6	100	-

Notes:

1. Sensing resistance in air (R_0) is measured under controlled ambient conditions, i.e. synthetic air at 23 ± 5 °C and <5 ± 5% RH.
2. Sensitivity factor (S_R) is defined as R_S at 0.25 ppm of NO_2 divided by R_S in air. Test conditions are 23 ± 5 °C and <5 ± 5% RH.

e2v

MiCS-2610
O_3 Sensor

This datasheet describes the use of the MiCS-2610 in ozone detection applications. The package and the mode of operation described in this document target the detection of the oxidising gas O_3 in indoor or outdoor environments. Ozone is a hazardous gas, which can cause respiratory problems at concentrations above 100 ppb.

FEATURES

- Low heater current
- Wide detection range
- High sensitivity
- Fast thermal response
- Miniature dimensions
- High resistance to shocks and vibrations

IMPORTANT PRECAUTIONS

Read the following instructions carefully before using the MiCS-2610 sensor described in this document to avoid erroneous readings and to prevent the device from permanent damage.

- The sensor must not be wave soldered without protection, or exposed to high concentrations of organic solvents, ammonia, or silicone vapours, to avoid poisoning the sensitive layer.
- Heating powers above the maximum rating of 95 mW can destroy the sensor due to overheating.
- After exposing the sensor to high concentrations of O_3, make sure the sensor is given enough time to recover before taking new measurements.
- For any additional questions, email enquiries@e2v.com or telephone +44 (0)1245 493493.

OPERATING MODE

The recommended mode of operation is a constant voltage mode. A heater power of P_H = 80 mW is applied. This causes the temperature of the sensing resistor (R_S) to reach about 430 °C.

Detection of the O_3 concentration is achieved by measuring the sensing resistor R_S during operation.

SENSOR RESPONSE

The sensor response to O_3 in air is represented in Fig. 1. The sensor resistance R_S is normalised to the resistance under 100 ppb of O_3 (R_{100ppb}).

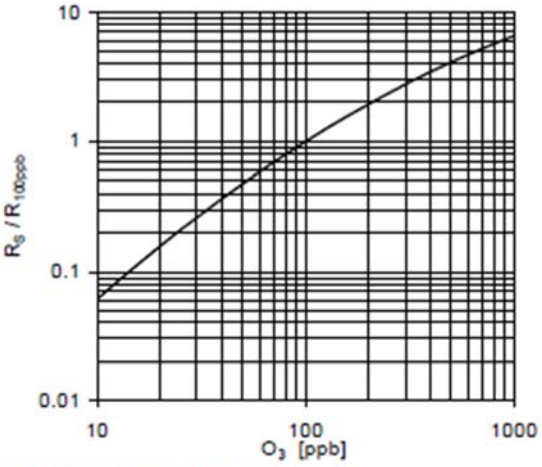
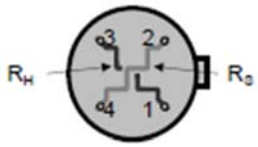


Fig. 1: R_S/R_0 as a function of gas concentration at 50% RH and 25 °C.

MEASUREMENT CIRCUIT

Fig. 2 shows the pin connections of the MiCS-2610 ozone sensor. A simple circuit to measure the O_3 concentration is proposed in Fig. 3. The heating voltage V_H is applied to pins 3 and 1. A load resistor R_L is connected in series with R_S to convert the resistance R_S to a voltage V_S between pins 2 and 4. R_S can then be calculated by the following expression:

$$R_S = R_L / (V_{CC} - V_S) \times V_S$$



Pin	Connection
1	Heater ground
2	Sensor pin
3	Heater power
4	Sensor pin

Fig. 2: Equivalent circuit of MiCS-2610 (top view)

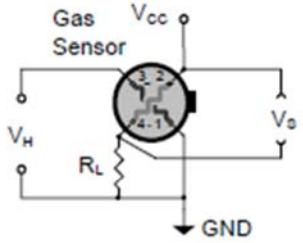


Fig. 3: Measurement circuit for O_3 detection

ELECTRICAL SPECIFICATIONS

Maximum Ratings

Rating	Symbol	Value/ Range	Unit
Maximum sensor supply voltage	V_{CC}	5	V
Maximum heater power dissipation (see note 1)	P_H	95	mW
Maximum sensor power dissipation	P_S	1	mW
Relative humidity range	R_H	5 – 95	%RH
Ambient operating temperature	T_{amb}	-40 – 70	°C
Storage temperature range (see note 2)	T_{sto}	-40 – 50	°C
Storage humidity range	RH_{sto}	5 – 95	%RH

Operating Conditions

Parameter	Symbol	Typ	Min	Max	Unit
Heating power (see note 3)	P_H	80	66	95	mW
Heating voltage	V_H	2.35	-	-	V
Heating current	I_H	34	-	-	mA
Heating resistance (see note 4)	R_H	68	58	78	Ω

Sensitivity Characteristics

Characteristic	Symbol	Typ	Min	Max	Unit
O_3 detection range	FS		10	1000	ppb
Sensing resistance in air	R_0	11	3	60	kΩ
Sensitivity factor (see note 5)	S_R	2	1.5	4	-

Notes:

1. Heating powers above 95 mW can cause permanent damage to the sensor due to overheating.
2. Storage of parts in original shipping package.
3. To ensure a correct operating temperature, the heater voltage should be ajusted so that the resulting heating power equals 80 mW. Lower heating power will reduce the sensitivity and increase the response time. Heating powers above 95 mW can cause permanent damage to the sensor due to overheating.
4. Heating resistor values from sensors out of production range between 58 and 78 Ω measured at V_H = 2.35 V. Due to material properties of the heating resistor its value increases during operating life.
5. Sensitivity factor S_R is defined as R_S at 100 ppb of O_3 divided by R_S at 50 ppb of O_3. Test conditions are 50 ± 5% RH and 25 ± 2 °C.

Temperature Sensor

AVERAGE AND DIFFERENTIAL TEMPERATURE MEASUREMENT

In many commercial and industrial environments, temperature sensors often measure the average temperature in a building, or the difference in temperature between two locations on a factory floor or in an industrial process. The circuits in Figure 28 and Figure 29 demonstrate an inexpensive approach to average and differential temperature measurement.

In Figure 28, an OP193 sums the outputs of three temperature sensors to produce an output voltage scaled by 10 mV/°C that represents the average temperature at three locations. The circuit can be extended to include as many temperature sensors as required as long as the transfer equation of the circuit is maintained. In this application, it is recommended that one temperature sensor type be used throughout the circuit; otherwise, the output voltage of the circuit cannot produce an accurate reading of the various ambient conditions.

The circuit in Figure 29 illustrates how a pair of TMP3x sensors used with an OP193 configured as a difference amplifier can read the difference in temperature between two locations. In these applications, it is always possible that one temperature sensor is reading a temperature below that of the other sensor. To accommodate this condition, the output of the OP193 is offset to a voltage at one-half the supply via R5 and R6. Thus, the output voltage of the circuit is measured relative to this point, as shown in Figure 29. Using the TMP36, the output voltage of the circuit is scaled by 10 mV/°C. To minimize the error in the difference between the two measured temperatures, a common, readily available thin-film resistor network is used for R1 to R4.

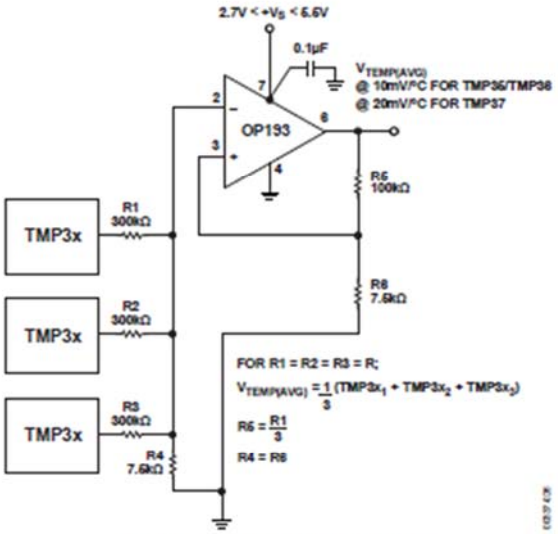


Figure 28. Configuring Multiple Sensors for Average Temperature Measurements

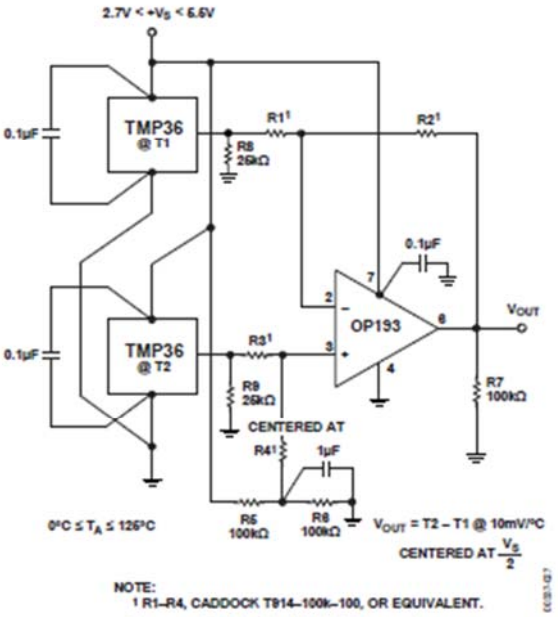


Figure 29. Configuring Multiple Sensors for Differential Temperature Measurements

OUTLINE DIMENSIONS

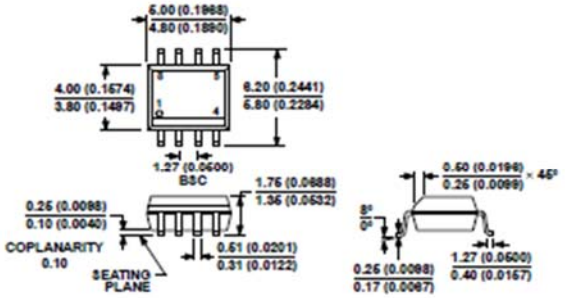


Figure 36. 8-Lead Standard Small Outline Package [SOIC_N]
Narrow Body
(R-8)
Dimensions shown in millimeters and (inches)

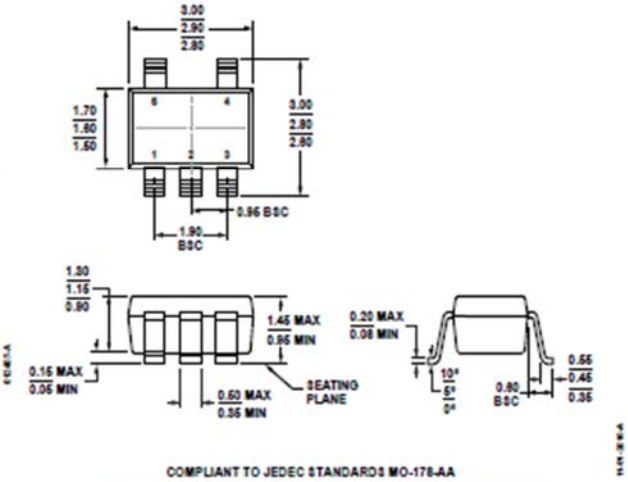


Figure 37. 5-Lead Small Outline Transistor Package [SOT-23]
(RJ-5)
Dimensions shown in millimeters

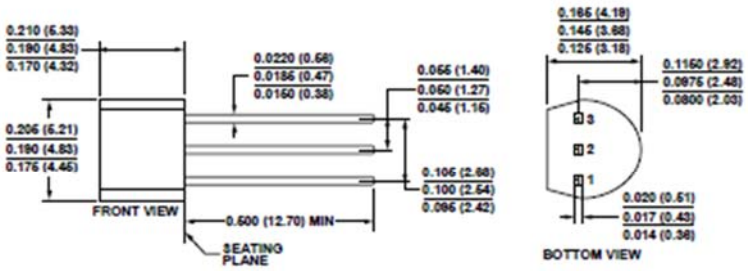


Figure 38. 3-Pin Plastic Header-Style Package [TO-92]
(T-3)
Dimensions shown in inches and (millimeters)

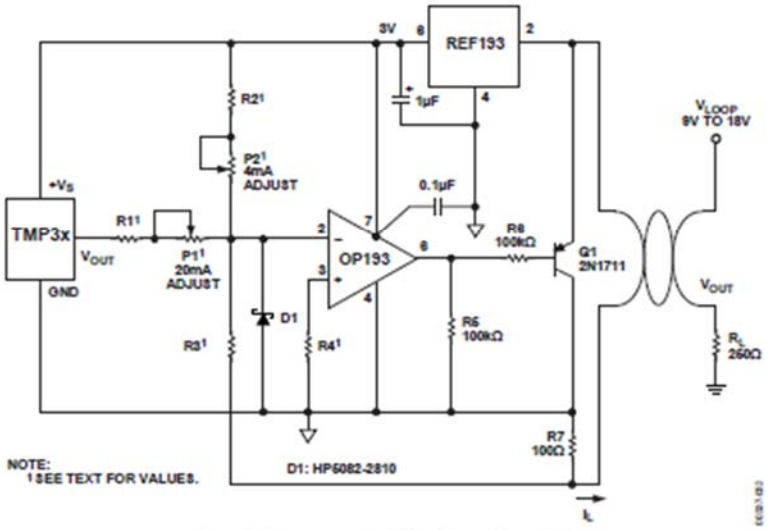


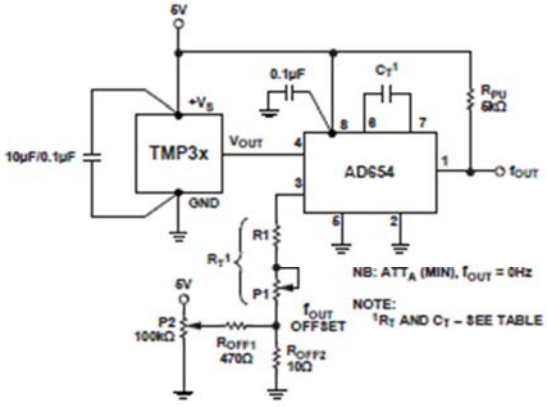
Figure 33. Temperature to 4–20 mA Loop Transmitter

TEMPERATURE-TO-FREQUENCY CONVERTER

Another common method of transmitting analog information from a remote location is to convert a voltage to an equivalent value in the frequency domain. This is readily done with any of the low cost, monolithic voltage-to-frequency converters (VFCs) available. These VFCs feature a robust, open-collector output transistor for easy interfacing to digital circuitry. The digital signal produced by the VFC is less susceptible to contamination from external noise sources and line voltage drops because the only important information is the frequency of the digital signal. When the conversions between temperature and frequency are done accurately, the temperature data from the sensors can be reliably transmitted.

The circuit in Figure 34 illustrates a method by which the outputs of these temperature sensors can be converted to a frequency using the AD654. The output signal of the AD654 is a square wave that is proportional to the dc input voltage across Pin 4 and Pin 3. The transfer equation of the circuit is given by

$$f_{OUT} = \left\{ \frac{V_{TPM} - V_{OFFSET}}{10 \times (R_T \times C_T)} \right\}$$



SENSOR	R_T (R1 + P1)	C_T
TMP35	11.8kΩ + 500Ω	1.7nF
TMP36	16.2kΩ + 500Ω	1.8nF
TMP37	18.2kΩ + 1kΩ	2.1nF

Figure 34. Temperature-to-Frequency Converter

Humidity Sensor

FIGURE 5. HIH-4030 MOUNTING DIMENSIONS (For reference only. mm/[in])

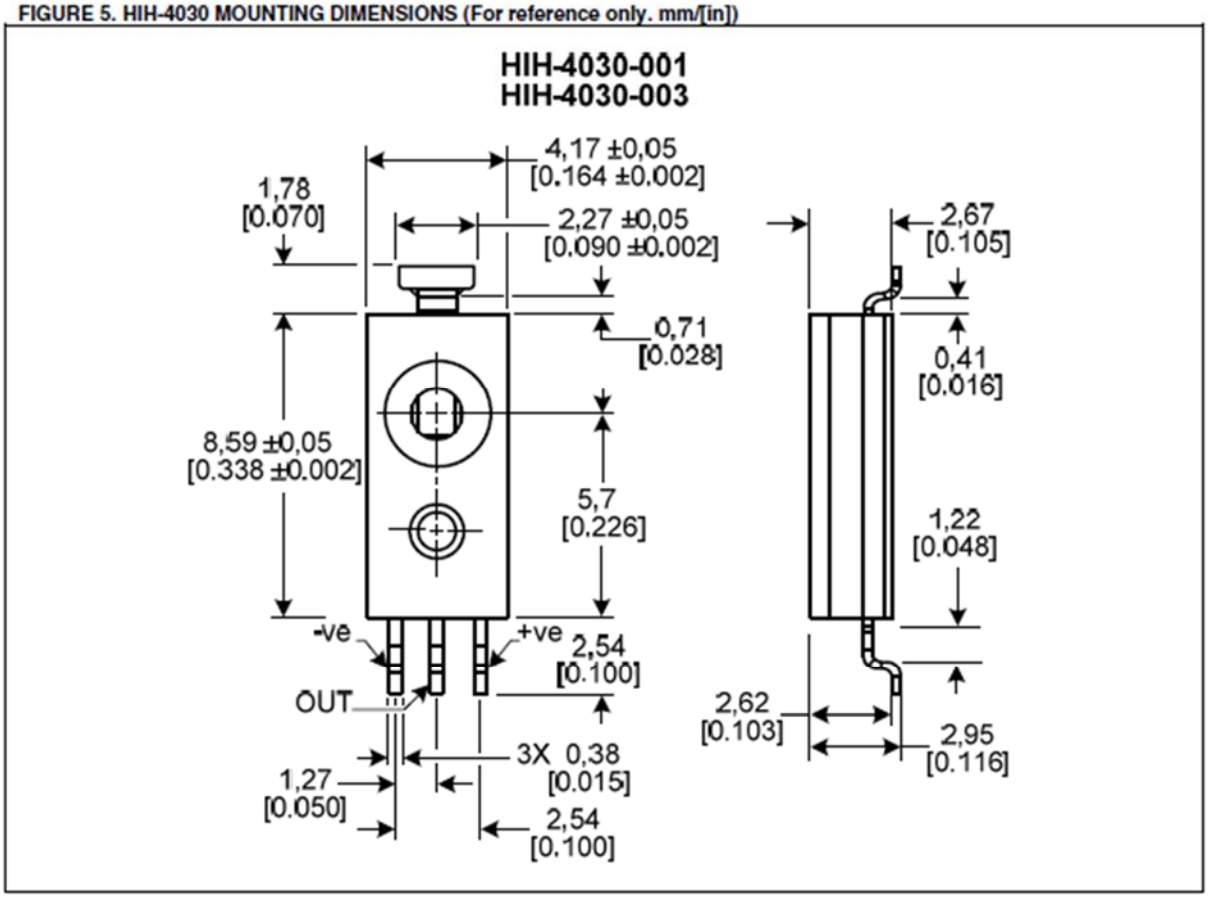


TABLE 1. PERFORMANCE SPECIFICATIONS (At 5 Vdc supply and 25 °C [77 °F] unless otherwise noted.)

Parameter	Minimum	Typical	Maximum	Unit	Specific Note
Interchangeability (first order curve)	–	–	–	–	–
0% RH to 59% RH	-5	–	5	% RH	–
60% RH to 100% RH	-8	–	8	% RH	–
Accuracy (best fit straight line)	-3.5	–	+3.5	% RH	1
Hysterisis	–	3	–	% RH	–
Repeatability	–	±0.5	–	% RH	–
Settling time	–	–	70	ms	–
Response time (1/e in slow moving air)	–	5	–	s	–
Stability (at 50% RH in a year)	–	±1.2	–	% RH	2
Stability (at 50% RH in a year)	–	±0.5	–	% RH	3
Voltage supply	4	–	5.8	Vdc	4
Current supply	–	200	500	µA	–
Voltage output (1st order curve fit)	$V_{OUT}=(V_{SUPPLY})(0.0062(\text{sensor RH}) + 0.16)$, typical at 25 °C				
Temperature compensation	True RH = (Sensor RH)/(1.0546 – 0.00216T), T in °C				
Output voltage temp. coefficient at 50% RH, 5 V	–	-4	–	mV/°C	–
Operating temperature	-40[-40]	See Figure 1.	85[185]	°C[°F]	–
Operating humidity (HIH-4030)	0	See Figure 1.	100	% RH	5
Operating humidity (HIH-4031)	0	See Figure 1.	100	% RH	–
Storage temperature	-50[-58]	–	125[257]	°C[°F]	–
Storage humidity	See Figure 2.			% RH	5

Specific Notes:

1. Can only be achieved with the supplied slope and offset. For HIH-4030/31-003 catalog listings only.
2. Includes testing outside of recommended operating zone.
3. Includes testing for recommended operating zone only.
4. Device is calibrated at 5 Vdc and 25 °C.
5. Non-condensing environment. When liquid water falls on the humidity sensor die, output goes to a low rail condition indicating no humidity.

General Notes:

- Sensor is ratiometric to supply voltage.
- Extended exposure to ≥90% RH causes a reversible shift of 3% RH.
- Sensor is light sensitive. For best performance, shield sensor from bright light.

FACTORY CALIBRATION DATA

HIH-4030/31 Sensors may be ordered with a calibration and data printout. See Table 2 and the order guide on the back page.

TABLE 2. EXAMPLE DATA PRINTOUT

Model	HIH-4030-003
Channel	92
Wafer	030996M
MRP	337313
Calculated values at 5 V V_{OUT} at 0% RH V_{OUT} at 75.3% RH	 0.958 V 3.268 V
Linear output for 3.5% RH accuracy at 25 °C Zero offset Slope Sensor RH	 0.958 V 30.680 mV/%RH (V_{OUT} - zero offset)/slope (V_{OUT} - 0.958)/0.0307
Ratiometric response for 0% RH to 100% RH V_{OUT}	 V_{SUPPLY} (0.1915 to 0.8130)

Battery

Portable Power Bank External 2600mAh Mobile USB

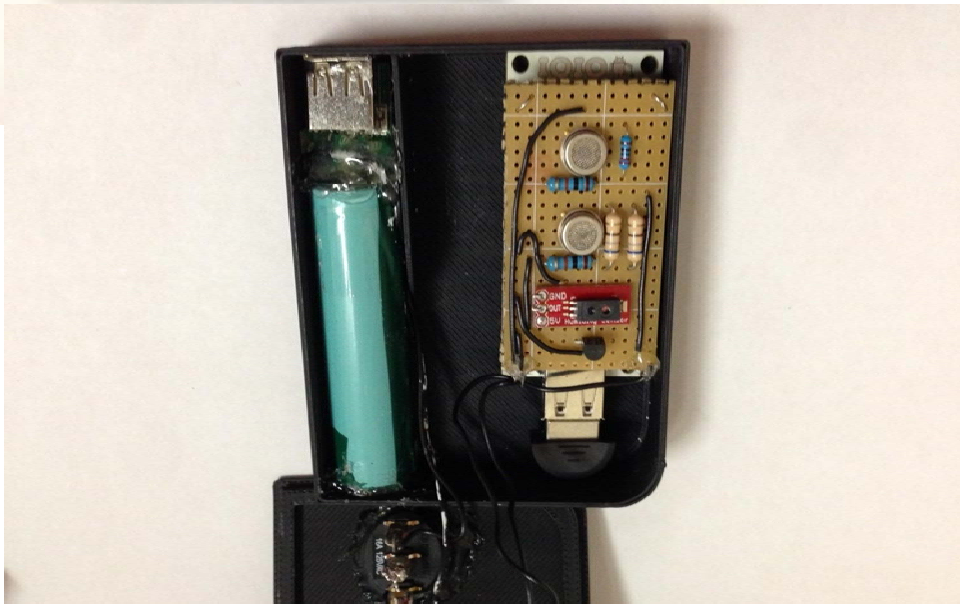
Parameter Name	Value
Capacity	2600mAh
Output	5V / 1A
Input	5V / 1A
Estimated Battery Life	Approx. 13 hours

Microprocessor

Microchip PIC24FJ256DA206-I/PT

Parameter Name	Value
Architecture	16-bit
CPU Speed (MIPS)	16
Memory Type	Flash
Program Memory (KB)	256
RAM Bytes	98,304
Temperature Range C	-40 to 85
Operating Voltage Range (V)	2.2 to 3.6
I/O Pins	52
Pin Count	64
Internal Oscillator	8 MHz, 32 kHz

Appendix 7.3 Physical Description of Product

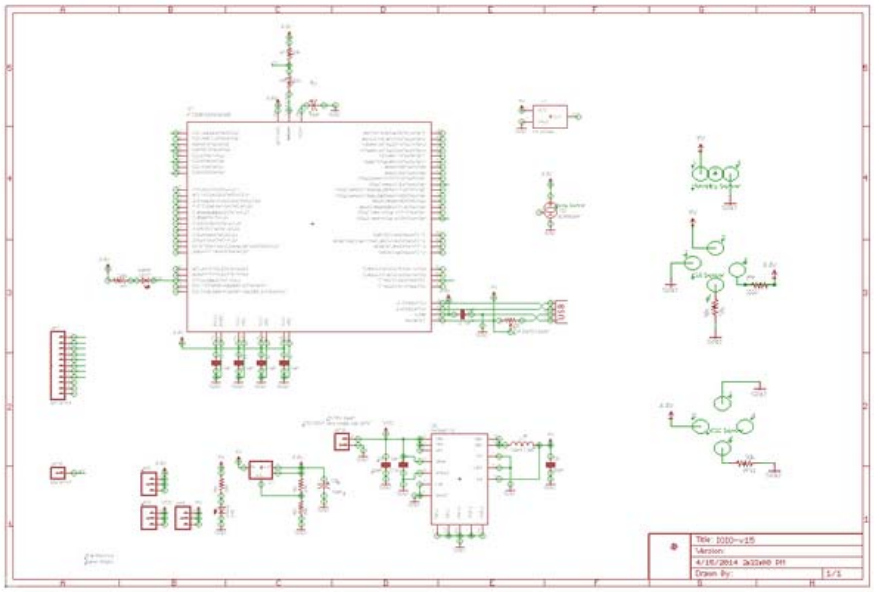


8. PCB Manufacturing

To reduce the physical footprint of the Air Quality Monitor device, the team decided that it would be best to reduce the size of the electrical circuit that stored all of the device peripherals. Nick and Brett were tasked with designing and manufacturing a circuit board that housed all electrical components. This would allow the device to go from having two separate boards, one containing the sensors and their peripherals, and the other containing the microprocessor (IOIO board).

8.1 PCB Schematic Design

The first step in the process would be to create a schematic on one sheet in a CAD electrical engineering software. The software Nick and Brett tried to familiarize themselves with first was ExpressSCH – the design software from ExpressPCB, a company that specializes in PCB manufacturing utilizing their software. After some work with this software and other searching, it was found that the best tool to use would be Eagle, primarily because the IOIO schematic was available online, in an open-source setting that would allow the team to import a large portion of the elements in the electrical design. Using Eagle CAD software provided a great time saving in that it allowed the team to skip the process of creating many of the symbols for the microprocessor and its peripherals that weren't in ExpressSCH's library.



Air Quality Monitor Schematic (In Eagle)

In the EagleCAD software, Nick and Brett were able to start with the IOIO schematic to keep important components from the development board, while reducing the size and removing other components not necessary for the Air Quality Monitor to operate. Components kept were the Step-Down Converter, SOT223 Semiconductor, Inductor, and TRIMPOT. Components that were removed were two 1x11 pin-through connectors, GND pin-through matrix, and various capacitors, resistors, and unused pin-through holes connected to pins in the microprocessor. This allowed for ample use of

the board and the ability to add and move other things to accommodate the sensors. It took a great deal of research to determine what each component did and if it was necessary for the Air Quality Monitor.

Once the board had all the proper components kept and removed, it was time to design it to the group's specifications. The first step was to remove all connections to the microprocessor. This was so the connections could be set up based upon the Air Quality Monitor's design, including the one 1x11 pin-through connector that was already connected pin-by-pin to the microprocessor. This was for multiple reasons, one being so that the pins can be designated however the group chose. The other was due to spacing, which will be referred to later in the report. Also, certain pins on the microprocessor could not be removed or moved to different pins due to the already embedded code that cannot be accessed by the users. This include the 4 USB connection power, and ground pins.

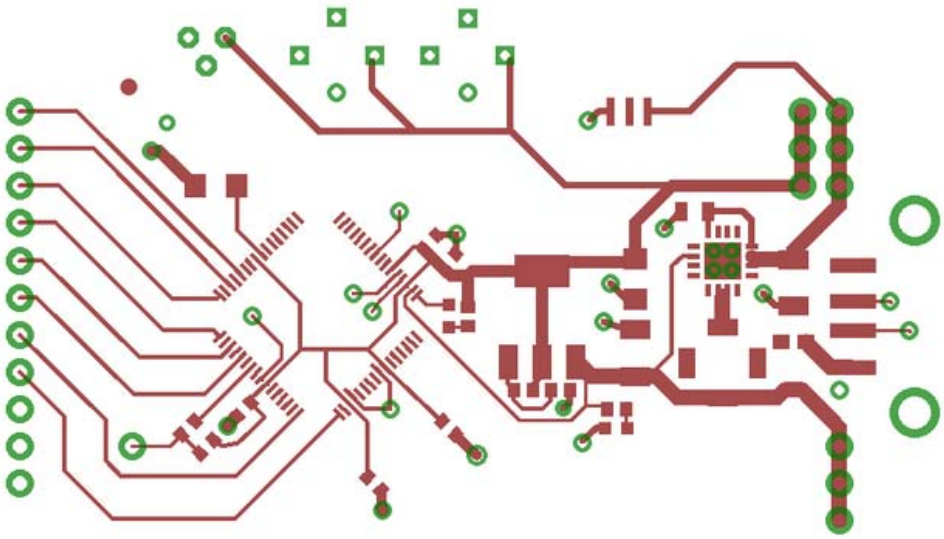
The next step was to draw out the sensors in the schematic. There exists a method of drawing out new components to be used in Eagle. Due to difficulty and time of actually making the components, the sensors were not fully incorporated into the schematic, but instead "represented." Instead, they were drawn in and connected as needed, but the Eagle sees these as electrical errors.

Once the schematic had been laid out and connections had been made, things were moved around in an aesthetically pleasing and easy to understand layout. It was time to begin the most tedious part of circuit design, PCB layout.

8.2 PCB Layout

The PCB Layout process is not a simple one. First, the designer must decide how many layers are to be used in the PCB. Some of the limitations involved in deciding how many layers to use include cost, company, need, number of components, space, and more. Because the free version of Eagle limited the designer to one schematic sheet and a two-layer board, it was sort of pre-determined that a two-layer board was going to be used in the layout process. The Eagle Lite software also limits the user to a 4x6 board size, but that was not a concern in this project.

Air Quality Monitor PCB Layout for Top Layer



With the number of layers determined, the next step was to export our

schematic design from Eagle's schematic creator to its PCB layout tool. This is a simple task, but the algorithm for laying out parts may be a bit outdated, so the placement of the components had to be done by hand. Most of the components kept from the original IOIO board were kept in the same place with the same traces, but others were moved for both spacing and accommodation.

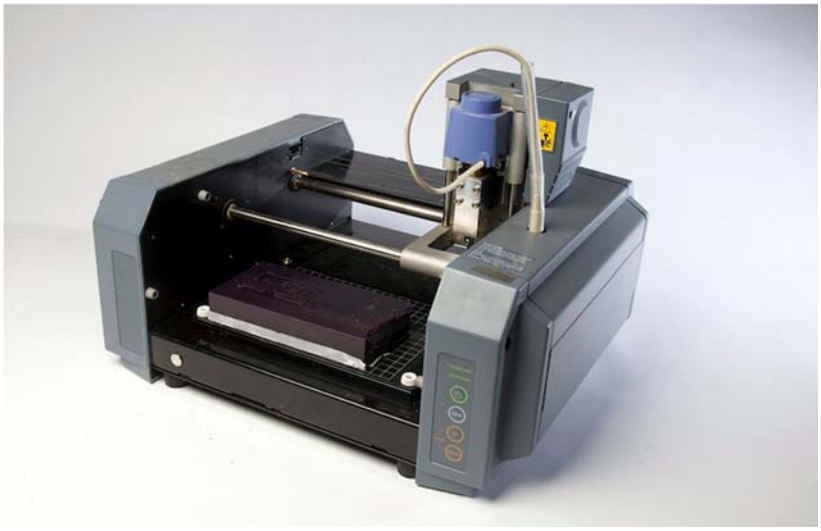
Once the placing was finished, it was time to lay traces out for the connections to be made from the sensors and their peripherals to the microprocessor. Eagle's layout tool has the ability to auto-route the traces in the best manner, according to its algorithm, but we found it to be unhelpful with what we were attempting to do. Our first step was to route connections to the sensors from their respective voltage needs. The top of the board can be seen to have 3.3V and VCC (5V). These were routed to the sensors as needed. We also needed to manually route the traces from the 1x11 pin-through connectors to the microprocessor pins. This was also done manually to make sure the spacing was adequate for the drill. Also, the polygon tool was used to draw out a ground plane on the bottom of the board. This would allow for the ground pins on any component to simply connect to the bottom of the board, which simplifies the layour and reduces traces needed. Once the schematic and layout of the device were finished, it was time to begin the manufacturing process.

Many modifications needed to be made during the PCB layout process, which may be referred to later. These included retracing of the pins, redrawing of the sensors, moving components, restructuring of the board, changing layers for routes, and removal of microprocessor pins. These were all done in response to errors encountered during the manufacturing process. The reasons for them will be addressed later.

8.3 PCB Manufacturing Process

The manufacturing process began with the team deciding where and how it would attempt to have a PCB manufactured. The two primary options were having Advanced Circuits make a few two-layer boards for us to purchase (2-layer boards are ~$33/each for students) and use for trouble-shooting, and the second option was to use the Fabrication Laboratory (FabLab) on Stout's campus to create our own 2-layer PCB.

Advances Circuits would send professionally done boards complete with solder mask and finish. The student price for these would again be $33 per board. They also used Eagle software as their primary PCB design software. The



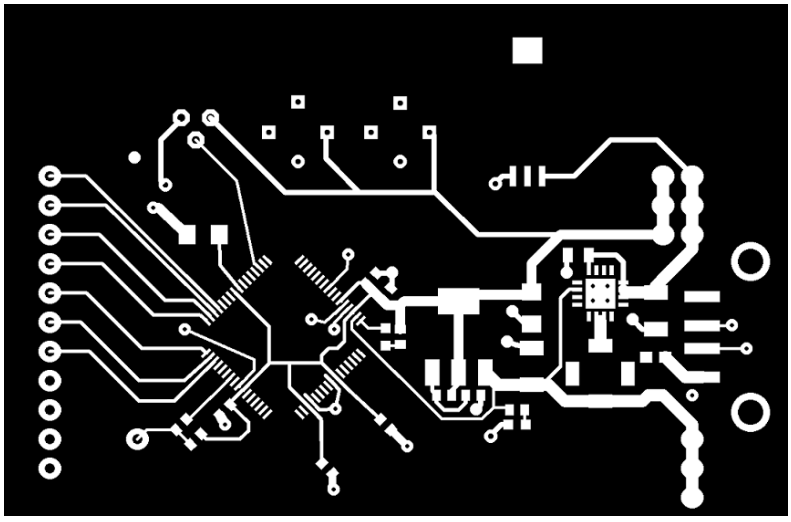
quote from their company put the time to get the boards at about 2-3 weeks

The FABLAB at Stout has a milling machine that takes in a schematic and cuts it out. The FABLAB director explained that we would not need to pay for any materials used since we were a part of the school. He also explained to us that not only can we do a two layer board, but it can be cut out in less than an hour.

Based on the cost-benefit ratio of comparing ordering boards from Advanced Circuits and using the FABLAB on campus, the decision was made to use the FABLAB. It should also be noted that doing the project in house adds to the accomplishments of both the Capstone classes and the FABLAB itself. The next task was then to go and manufacture a printed circuit board.

This task proved to be very difficult over the time that Nick and Brett spent in the FabLab attempting to get a functional PCB. The board cutting process involves as follows:

- Export all layers (Top, Bottom, Holes) as high resolution monochrome images
- Touch up any pixel problems using an image editor
- Invert the colors of the board to match the program's needs
- Take the images to the Linux machine and run fab in the terminal
- Select Roland Modela and the type of .png
- In the software, upload one of the images being used (typically top first)
- Change drill settings which included: Drill size, Trace size, number of paths around, x/y coordinates, z depth, overlap, and speed
- Make the path and check to make sure it is clean
- Make the .rml file and send it to the drill



Monochrome image export

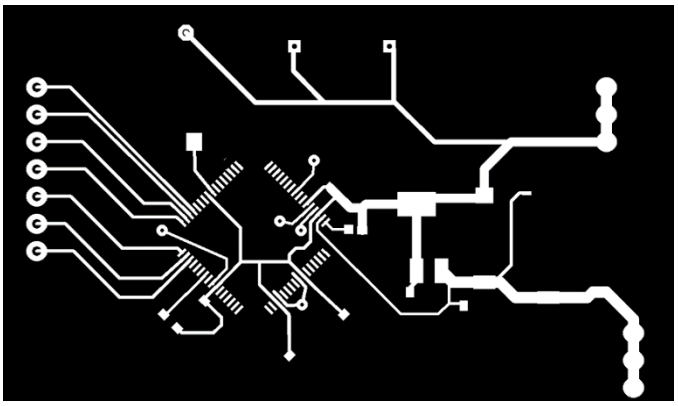
Initially, we attempted to cut the board using one setting and the 1/64" drill bit. This worked great for most of the board. The problem was that the drill bit was simply too big to get into the microprocessor and some of the traces that were associated with that small of a spacing. The first attempt to try and reduce the overlap to see if it could fit into those spaces ended up destroying the pads for the microprocessor.

Our second attempt was done with the same settings, but now we had a 10/1000." The board was cut except for the route spacing smaller than the 1/64" drill bit

(about .4 mm). The drill bit was changed and the settings kept the same. This was basically doing the same route as the first drill bit, but the drill size was reduced from .4mm to .25mm. Making the path in Roland Modela proved to show that it could indeed get in between the microprocessor pins. The path was run and found that at the normal speed and the normal about of paths around (4), running it again destroyed many routes, mostly to the 1x11 pin-through connectors.

The next attempt was when changing to the smaller drill bit; reduce the speed and the paths around. However, due to other students using the machine, we found that the board table was not balanced. This caused parts of the board to be deep cuts and others to be very shallow. When the z depth is too shallow, it will shred the copper on top, causing poor or shorted routes.

Once the table was leveled another attempt was made. The 1/64" cut as expected. The drill bit was changed to the 10/1000," the speed and paths were reduced and ran. After it finished and upon examining the board, we found that even with reduced settings, certain traces would still get cut or mangled in the process. Because of the small size of the drill, any very small variations would cause problems such as these.



Reduction of Drill Cutting

It was at this point it was decided to go back to the layout process, as stated earlier. At this point, we reduced the amount the small drill bit needed to cut by modifying the exported images. This was done by removing anything that was not directly connected to the microprocessor. This allowed us to reduce the .25mm drill bit's cutting by about 60%. Upon running the program again, we were able to get in between the microprocessor pads, but the traces to the 1x11 were still problematic in the end.

Again, it was back to the design process again. The board layout was modified to have larger traces to the 1x11 connectors as well as change the spacing to allow the maximum amount of space between traces. This would have worked, except the z depth was changed to be too deep, which destroyed the pads on the microprocessor. Another attempt was made with a better z depth, and upon initial examination, seemed to be useable. So at this point, the board was flipped over and the bottom image was inverted, mirrored, and uploaded for cutting. The bottom was cut and it was found that it did not line up with the top. Some of the holes would have directly taken out routes or pads. The problem was that the spaces from the edge were different on the bottom and the top.

This was changed in Eagle, double checked by comparing and overlapping the images exported, and seemed to match up. These were taken to the drill and this time it was the bottom that was cut out first. We found that after running on the bottom, the x/y coordinates did not match up with (0, 0) on the actual table. This took some tampering to be able to reset the drill's (0, 0) spot to the correct spot. It was attempted again and seemed to work. The board was flipped correctly for the top. The 1/64" drill ran and cut out correctly. The 10/1000" drill bit was replaces, the settings reduced, and the image edited to reduce the amount of cutting done by the small bit. It ran and almost cut out everything correctly. The problem was even with the small cutter, some of the pins and traces simply could not be cut even by the smaller cutter. Along with this, some of the pads were taken off by the small bit.

Please insert front and back picture of one PCB here

Due to the problems encountered, we decided to pursue the other initial decision of getting boards printed from Advanced Circuits. However, the department and the Capstone class encountered budgetary concerns that disabled the group from being able to order boards.

A few more attempts were made to cut out a board based upon the last try, but with trying to change certain drill settings to help alleviate the problems encountered. These again turned out to be unsuccessful. At this point, it was realized that with a full printed circuit board and the small size of the pads on the microprocessor, there were simply too many variables to be considered based on the amount of time left in the semester.

In all, Nick and Brett attempted over ten different times (Each attempt taking from 0.5 to 6 hours at the manufacturing station) to create a 2-layer PCB using this machine, and each time something different went wrong and had to be changed, in hopes of having the next attempt work. To summarize all the problems encountered:

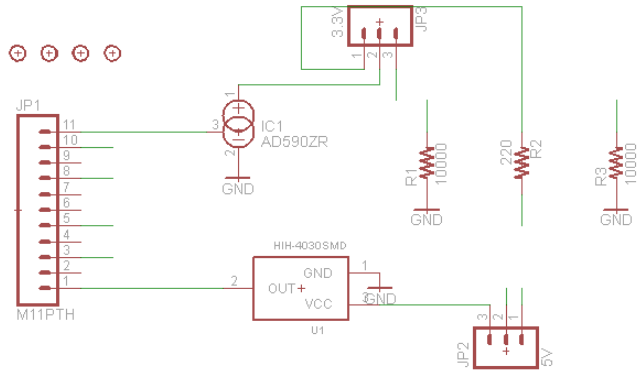
- Drill Settings off
- Leveling problems
- **Pads and routes too small**
- Layout errors
- Drill defects and errors
- User error
- Layer lining up problems
- Shredding of routes
- Drill x/y off with program

Eventually, the team decided it was best to move forward with the other of the original ideas, which was having the PCB manufactured by Advanced Circuits. Due to a lack of budget, the team was unable to pursue this route, and so up came the idea of

creating a break-out-board to place the sensors and their peripherals on, and connect it to the IOIO board.

8.4 Breakout Board

To try and get a working product in the short time left in the semester, Nick and Brett pursued the idea of a breakout board. The breakout board consisted of the sensors and any peripherals needed to run those on a printed circuit board that would connect to the IOIO board. This would basically sit on top of the IOIO development board and still help to reduce the size of the Air Quality Monitor.

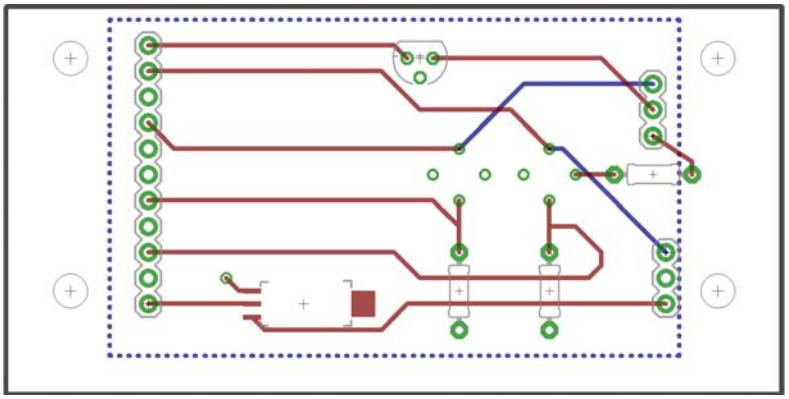


Breakout Board 1 Schematic

The breakout board was designed based upon strictly the sensors and any extra components needed for them to run. This also included the 1x11 pin-through connectors so that it could line up with the IOIO board with ease for input purposes.

The PCB layout itself was fairly simple. The sensors were connected to power, ground and the 1x11 connector as needed, along with peripherals. The routes were made more than large enough for the drill to be able to cut them with ease. The Polygon tool was again used to create a ground plane on the bottom of the board.

Nick and Brett then did the same process as before (PCB schematic and layout design) and attempted to manufacture the break-out-board as a two-layer board. With about two weeks left before the final presentation, Nick and Brett began the work and were able to complete the milling of the board and soldering of components onto the break-out-board one day before final presentations. This proved to be insufficient time to be able to implement the software changes and make the connections before the final presentation, so the break-out-board still stands as an untested prototype.



The soldering of the two-sided copper boards (Standard for the FabLab, size: 2x3") is very complicated. Because the copper is extremely thin and lay on a tape, it is easily pulled from the

fiberglass middle when its temperature rises. In the soldering process of the sensors and peripherals to the break-out-board, the technician had to be very careful with how much solder was used and how much heat was applied to the copper pads. If the technician wasn't careful, they could lift the copper from the fiberglass and cause a solder-bridge to occur, which is undesirable in maintaining signal integrity.

With the breakout board completed, we used a Digital Multi-Meter to check if all the traces and routes for signal integrity. They turned out to all be of correct electrical properties. However, the group could not meet to work out all the kinks with the breakout board to get it in the Air Quality Monitor. It was estimated that one more week would have sufficed to get the breakout board integrated in.

Please insert Front/Back Picture of soldered Breakout board here

8.5 Next Step

In order for the continuation of this project to be successful, the team fully believes that it is in the best interest to pursue a third party to have a PCB manufactured. The PCB milling machine proved to have too many flaws to create a board with such small (6 mil) traces and spaces between traces. Some companies that do PCB manufacturing, and even offer discounts for students, include Advanced Circuits and PCBExpress. It is recommended that the next group contact these companies, requesting quotes, as soon as possible in the team's work. Knowing the budget and what process is best right away will greatly reduce surprises at the end, both financially and in accordance with time.

Having a PCB manufactured will not only allow for a more precise, accurate board to be made, but it will allow for less guessing in the trouble-shooting process. Additionally, having a PCB manufactured will significantly decrease the footprint of the 'guts' of the project, so to speak – because the mechanical footprint will be much smaller by having a single board contain all of the components and elements.

Project Patient Helper Beta

The Current State of Spatial Data Recording, Processing, and Storage

Eric Bonsness

Spring 2014

I. INTRODUCTION

Patient Helper Beta is an Android application that is targeted for asthma sufferers. Its purpose is to record environmental and spatial data of patients for future analysis. The app also allows the patient to record information about any attacks that he/she has while recording the other variables. The app does have other functions that are either implemented or in the development process. For more information on those features, refer to the documentation provided by Justin Cramm, Chase Meyers, or Jamieson Lutz, the other three members of the development team. However, for the purposes of this document, I will concentrate on the features that I have modified or developed. These features are related to the recording of the spatial data and storing it on the server.

Prior to my involvement with the project, the app recorded the spatial data into a local buffer that gets uploaded to the server when either the recording timer reaches its preset time limit, or the user presses the submit button. When uploaded to the server, the PHP code would assign values to the appropriate XML tags and append the data to a single text file. My job was to modify the type of file that is stored and allow for multiple files to be stored in a file hierarchy with unique directory names.

II. RESOURCES

The files for the Patient Helper Beta app can be found on the server redcedar3 in the directory /var/www/html/testSite. There are some files located in the directory that were created by one of the original developers of the app, Ben Koerner. However, some of them are not for this app and I don't even know their purpose. In fact, those files could probably be deleted. The files and folders in this directory for the app are as follows:

android – This folder is where the source code for the current version of the app is stored. Here you will find all the files for the Eclipse Android project.

testMaps – This is the Eclipse Android project stored on the server in the android directory . In this project, you'll find the Eclipse Android project source code for the entire app. The source code that I worked on and is referenced in this document are explained in the next three files.

SpatialGPSLoggerActivity.java – This is the Android activity source code that provides the user with a user interface to communicate with the service that may or may not be running in the background. It is located under testMaps/src/com/testMap.

SpatialGPSLoggerService.java – This is the Android service source code that performs the logging of the spatial data. It also is responsible for calling the method to post the data to the server. It is located under testMaps/src/edu/uwstout.

PostToServer.java – This is the source code for the class that connects to the server and sends the data. It is located under testMaps/src/koerner/benjamin.

gpsData.txt – This file is where the previous version of the app would append all the GPS data in XML tags from any device running the app. There are two related files that were kept that hold similar data; gpsData2.txt, gpsDataOld.txt.

processGPSData.php – This is the PHP file that handled the incoming GPS data on the previous version. This file appended the data into gpsData.txt.

processGPSDataTest.php – This is the PHP file that the current version uses to handle the incoming GPS data. This file is where the appropriate directory, named by device id, and filename is determined for storing the data. It returns a value of either the filename or false, if it fails. Another version of this file is processGPSDataTest1.php that was kept as a fallback in case the modifications did not work.

Directories that are series of random numbers and letters are actually named after the device id of the devices that are submitting data. Inside these directories are the KML files that hold the submitted GPS data and can be viewed with either Google Earth directly or Google Maps by entering the URL of the file on the server in the search bar. The URL to enter is in the form

http://redcedar3.uwstout.edu/testSite/directory name/KML file name

There may be some files that are named “false” or similar to this. They are products of failed data storage. I’m not exactly sure where the bug causing this error is located, but this is definitely something that should be addressed as high priority when the next team begins development. This issue is covered more extensively in the Future Work section of this document.

III. USER STORIES

Since both my user stories are very much related to each other, I'm going to avoid repeating myself by covering both of them in this section.

User Story 1 – As a developer, I would like the app to store the coordinate data in multiple KML files so that each log can be in its own file.

User Story 2 – As a developer, I would like to develop multiple KML files and develop the file hierarchy.

A. CONFIRMATION

1 – Show newly created files to show the data that was captured and stored in its own file.

2 – Show the file hierarchy on the server and show that new files go to their correct locations.

B. GENERAL DESCRIPTION

My deliverable artifact for this user story was the PHP code in the file processGPSDataTest.php, the file hierarchy, and the KML files stored on the server. My goal was to update the previous version from one that stores all the data in one big text file to one that has a file hierarchy with unique directory names and can also store different data recording sessions in their own unique KML files under the appropriate directory.

C. TECHNOLOGY

Java – The Java programming language is used for the development of the Android app. It is used to give the user interface of the app its functionality. It provides listeners for user interaction as well as process the data acquired in the activities and services.

PHP – The Hypertext Preprocessor language is the server-side language used to handle the data that is sent to the server. It is used to process the incoming data that is in the form of an associative array and output the data into KML formatted files in the correct directories.

KML – The Keyhole Markup Language uses a case-sensitive tag-based structure based on the XML standard. If you want to be technical, KML is actually XML. However, the elements and attributes are defined for use in viewing data in Google Earth.

XML – The eXtensible Markup Language is used in the creation of KML files and for building the display for Android applications.

D. SPECIFIC DESCRIPTION

The process starts with spatialGPSLoggerActivity.java. When a user selects the GPS tab located on the main screen of the patientHelper(Beta) app and then presses the start button, the spatialGPSLoggerActivity is started which also begins the spatialGPSLoggerService. The service records the spatial data as location data that is defined by the Android OS and stores it in a local buffer. When the user either presses the submit button or the timer reaches its pre-designated time limit, the GPS location data is converted into name-value pairs (associative array) along with the device id and current filename of the session, if a filename has been created. Initially, the filename is null until after the first submit and the server responds with the filename. The array is passed to the PostToServer class that actually connects to the file processGPSDataTest.php located on redcedar3 and posts the data.

When the server receives the data, it checks if there is a directory named by the device id. If no directory exists, is creates the directory and gives full permissions. Next, it checks the filename to see if it is null. If null, then it creates a file named by the current timestamp, sets its permissions to full, and writes the KML header and opening tags. If the filename is not null, then the script knows that it needs to append to the existing file, in which case it reads the current contents of the file into the $coords variable so that the new data can be appended to it and rewritten to the file. From here the spatial data enters a loop that extracts the desired data and appends it to $coords in KML format. When finished, $coords is written to the file and the filename is returned.

From here, the response from the server is passed back to the service from PostToServer, or PostToServer returns false if there is a connection failure. This is where I believe the source of several bugs stem from and how there can be some files on the server named false, as I previously mentioned. Once the service gets the response, the serverListener interface, which allows the service and the activity to communicate with each other, displays the response on the screen.

E. TESTING

1. PASSED

What has passed testing is format in which the data is sent to the server so that it can be processed properly. And, when the PHP receives the data, it will correctly format the data into the proper directories and KML files. Finally, the KML files are properly formatted so that they can be viewed in Google Earth. These are the basic tests that have passed to show that the functionality is working when everything is running smoothly. However, as we all know, smoothly doesn't always happen and many unknowns have reared their ugly heads during the testing that can make for some unpleasant results. See Future Work.

2. FUTURE

The only test that I can think of that was never really performed is multiple devices submitting data at the same time. This would test the server as well as the PHP code to see if it is efficient enough to handle a large surge of incoming data. Other tests that are related to the bugs that have occurred apply more to future work as they have more to do with how the application handles certain situations.

F. FUTURE WORK

1. LOST OR NO DATA CONNECTION

The application needs to have a way to deal with "no connection" or "lost connection" so that data does not get lost and submission errors are handled. Currently, the app does not check for data connection. If the submission fails, the data that was stored into a temporary array is just added back into the buffer. This changes the order in which the data was received into the buffer. Handling this, I believe, would solve the issue of having files with the name of false on the server.

2. STATIONARY DATA

If the user is stationary, the data keeps getting recorded. This can amount to large files of essentially useless data. In most cases, stationary data won't even make a mark in Google Earth when the KML file is viewed.

3. STORING ADDITIONAL DATA

The current version only stores the longitude, latitude, and altitude of the GPS data. Previously, it stored time, velocity, accuracy, and bearing. Currently, this data is recorded and sent to the server, but the PHP does not extract it. I did not get a chance to see if KML files can store this data. I believe it can, but the format of the KML files will need to be changed. This is information that Dr. Bae said needs to be recorded.

4. NOTIFICATIONS

There should be some sort of notification when the service is running in the background. We've found that users are not always aware that the service is still running or they thought they shut it down. This can be handled by the notification manager. Another possibility is to have some sort of timer that can pause, shut down, or prompt the user to shutdown the service after a lengthy runtime.

5. GPS SERVICES STOP WHEN PHONE SLEEPS

We found that when the user's phone goes to sleep, the GPS services on the phone that record location data go to sleep as well. This happens even if the service is running in the background. A possible remedy to this is to look into the PowerManager.WakeLock permissions in Android.

6. BATTERY MANAGER

This app can shorten battery life. Currently, there are no checks to handle when the battery is getting low. It's possible that Android handles this already, but it should be researched.

7. NAMING CONVENTIONS

It always seems like you give buttons and other UI features good, self-explanatory names when you're the developer. But, once a non-developer starts to use it, it becomes clear that the names are not self-explanatory. There has been some confusion as to what functions some of the buttons perform.

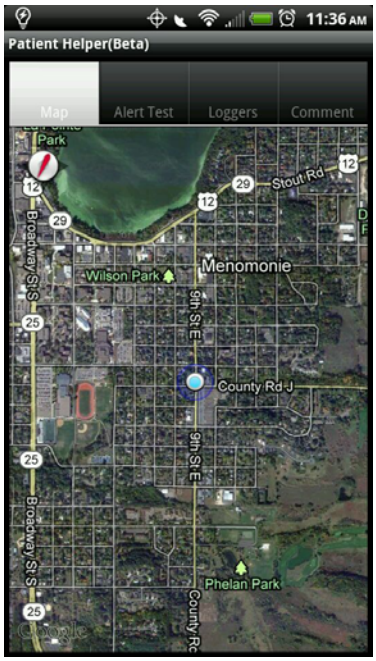
IV. CONCLUSION

I hope this document gives you future developers a much better idea of what you're getting into than what I had. FYI, I had no documentation to refer to and the code wasn't even commented. I did comment all the current and previous code for you. It doesn't go into fine detail, but it gives a developer a sense of what's happening. You're welcome. It's a little arrogant but oh well. If you are interested in programming for Android, this is a great project. It has features that will teach you about many different and cool Android functions and give you a deeper understanding of how Android works. Have fun and happy coding.

User Interface Design

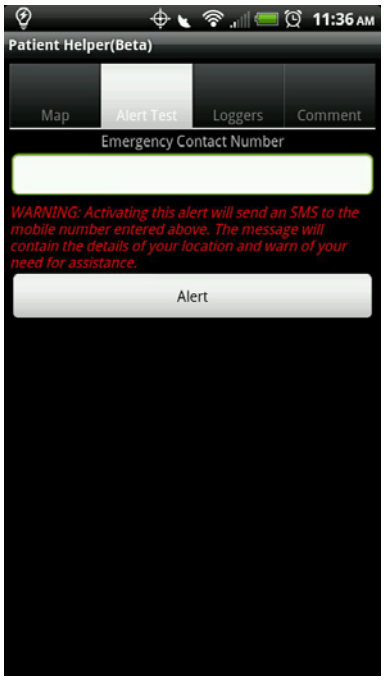
The Tab Layout

The application uses a tab layout to hold its many activities and function. The layout consists of four tabs (Map, Alert Test, Loggers, Comment).



The Map Tab contains the MapView that is run using the Google Maps API. The map overlay has a beacon of you current location and a compass.

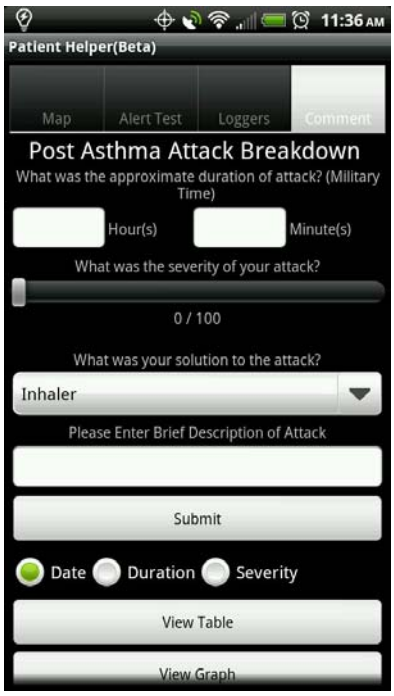
The Alert Test Tab contains a text field, for a emergence contact number, and a button that sends a text message that with your nearest street address.



The Loggers Tab consists of two buttons that launch the SpatialGPSLogger and the BatteryLogger.

The Comment Tab allows a user to comment on recent asthmatic attacks and also with logged comment. All of the comment data is stored in a local SQLite database.

When the application is paused the



onRetainNonConfigurationInstance method is called. This method saves the current tab that is highlighted in the layout. When the application calls the onCreate method again it will call the getLastNonConfigurationInstance method which returns the integer value that represents the tab stored above. These methods are important when updating the layout during the orientation changes of a device.

Google Maps MapView

The map view is updated by the onLocationChanged method. Upon a location change the location variable are updated. The GeoPoint and GeoCoder is updated, which is responsible for retrieving the street address. The map view is updated by calling the animateTo method which passes into the new location.

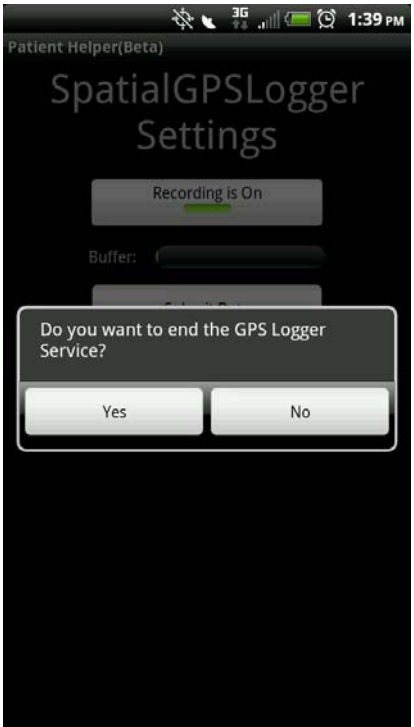
The onPause method disables the maps compass and the onResume methods enables the compass.

User Story 1:

As a User I would like to know if the application is still its services so that i can better manage my battery life.

1.1 Confirmation: When exiting the GPS Logger the user will be prompted with the opinion to close all services or not. Test each choice and then verify running services on the phone. Settings>Applications>Running Services: The location of your running services menu may vary depending on the phone.

1.2 Description: I implemented a onBackPress method that would create an alert dialog with two options for a response, positive or negative. With a negative response we would start the service and with an positive, stop the service.



1.3 Technology Used: onBackPressed(), AlertDialogBuilder class, DialogInterface.Listener class.

```
    public void onBackPressed() {
                AlertDialog.Builder builder =
new AlertDialog.Builder(this);
          builder.setMessage("Do you want to
end the GPS Logger Service?")
                 .setCancelable(false)
```

```
                .setPositiveButton("Yes", new
    DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                          stopService(new
    Intent(SpatialGPSLoggerActivity.this, SpatialGPSLoggerService.class));
                        SpatialGPSLoggerActivity.this.finish();

                    }
                })
                .setNegativeButton("No", new
    DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                                    startService(new
    Intent(SpatialGPSLoggerActivity.this, SpatialGPSLoggerService.class));
                                    SpatialGPSLoggerActivity.this.finish();


                    }
                });
            AlertDialog alert = builder.create();
            alert.show();

        }
```

1.4 Testing:

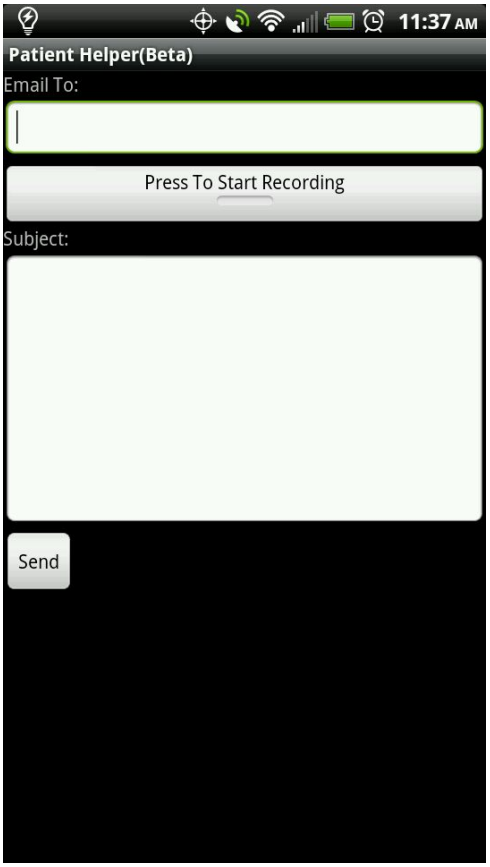
Step1:We checked the running services before starting the logger. The spatialGPSLogger service will NOT be running.
Step2: Start GPS Logger and return to the running services to verify the service has started.
Step3: Press Back. The Alert Dialog will be prompted and you will make a negative decision.
Check: the running services again to verify the service is still running.
Step4: Return to the SpatialGPSLogger and press back again. This time, respond positively.
Checking: refer to the running services, the SpatialGPSLogger is not be running.\

1.5 Future Work:

This method of service management needs to be implemented for every service that is created in the future, so that we can maintain consistence.

User Story 2:

1. As a developer, I would like to collect the battery usage information from a mobile device for statistical verification.



1.1 Confirmation: Demo application to show the collection of the battery information. Show collected data and view email.

1.2 Description The battery logger allows a developer to view and relay valuable battery levels. A developer can email this information to a recipient for further analysis.

1.3 Technology Used: When the battery logger button is pushed in the logger tab, a new batterylogger activity is created. Here you will see a the layout with a few button and text fields. Once you press the “Press to start Recording” button, the logger will record your battery level every 30 seconds. The Content in the large text field will continually update every 30 seconds. Once you have acquired a desired amount of information and the text field below “Email To:” has the recipients email address, press the send button. The Send button will then start an email intent. Verify the information and press send and the recipient will receive and email as describe

1.4

```
public class BatteryLoggerActivity extends Activity{

private TextView contentTxt;
private TextView emailTo;
private String[] recipients = {"address"};
private ToggleButton recordingButton;
private Calendar time;
private String formattedDate;
private int level;

// BroadcastReceiver that listens for the battery information to be received
  private BroadcastReceiver mBatInfoReceiver = new BroadcastReceiver(){
    @Override
    public void onReceive(Context arg0, Intent intent) {
      // TODO Auto-generated method stub
      if(recordingButton.isChecked()){
      int level = intent.getIntExtra("level", 0);
      time = Calendar.getInstance();
      SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
      formattedDate = df.format(time.getTime());
      contentTxt.setText(contentTxt.getText()+ "\n" + "Date: "+formattedDate+
"     " + "Battery Level: "+String.valueOf(level) + "%");
      }
    }
  };
```

```
//The onCreate Method define all of our attributes and parses them to our xml
file by using the findViewById method.
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
    setContentView(R.layout.batterymain);
    contentTxt = (TextView) this.findViewById(R.id.emailContent);
    contentTxt.setText(TestMapsActivity.getBatteryStats());
    emailTo = (TextView) this.findViewById(R.id.emailTo);
    this.registerReceiver(this.mBatInfoReceiver,
    new IntentFilter(Intent.ACTION_BATTERY_CHANGED));
    recordingButton = (ToggleButton) findViewById(R.id.toggleButton1);
      Button batteryButton = (Button) findViewById(R.id.send);
      batteryButton.setOnClickListener(new View.OnClickListener() {

//The onClick method listens for the send button to be push that starts the
new email intent activity and also passes in our fields into the new email
intent by using the putExtra method.
public void onClick(View v) {
// TODO Auto-generated method stub
try{
      recipients[0] = emailTo.getText().toString();
      Intent email = new Intent(Intent.ACTION_SEND);
      email.putExtra(Intent.EXTRA_EMAIL,recipients);
      email.putExtra(Intent.EXTRA_TEXT, contentTxt.getText());
      email.putExtra(Intent.EXTRA_SUBJECT, "Battery Logger Information");
      email.setType("message/rfc822");
      startActivity(email);
                                    }
      catch(Exception e){
      Toast.makeText(getApplicationContext(), e.toString(),
      Toast.LENGTH_SHORT).show();
     }
}
});
}
 //This will set out store our recorded information if the onPause method is
 called
      @Override
      protected void onPause() {
            // TODO Auto-generated method stub
            super.onPause();
           TestMapsActivity.setBatteryStats(contentTxt.getText().toString());

      }

    }
```

1.5 Testing:

Step 1: Start Activity

Step 2: Press the “Press to Start Recording Button”

Check: Notice the text field updates every 30 seconds.

Step 3: Fill in the text field below “Email to: ” with a valid email and press send.

Step 4: Verify information and press send again.

Check: Verity the email was sent to the email provided.

1.6 Future Work:

The logger needs to trigger a service so that the recording of the battery information can be continuous while not on the battery logger activity screen.

When the orientation of the device changes the onCreate method is called which triggers and the logger to update the battery level.

This Activity is for the developer and should be separate form the patient helper activity

We had planned to view the gathered battery information in a graph.