Oil Detection Sensor

Oil Detection Sensor

Major Qualifying Project Report completed for a Bachelor of Science in Electrical and Computer Engineering degree at
Worcester Polytechnic Institute, Worcester, MA

Submitted to: Professor John Orr

Submitted By:
Austin R. Alibozek
David H. James

April 30, 2015

Table of Contents

Acknowledgements:

We would like to thank the following people and departments for the support and help that they provided throughout the building, designing, and creating a prototype of our Oil Detection Sensor.

John Orr: Project Advisor and ECE Professor
Thank you for the help with ideas during the design phase, brainstorming solutions to problems that came up along the way, and providing us with helpful feedback throughout the project.

Joe St. Germain: RBE lab advisor
Thank you for the great tutorial on how to use UltiBoard so we were able to design and order the final PCB prototypes.

Robert Boisse: ECE shop
Thank you for all the help ordering all the needed parts for the project. It was also very helpful when it came time to solder the very small surface mounting chips and populating our PCB.

Department of Electrical and computer Engineering
We would just like to thank them for the opportunity and challenge with coming up with an idea of how to use a sensor in the usage of oil being burnt in a house hold that uses an oil burning furnace to heat their home during the winter.

We would also like to take a moment and thank everyone that went to the presentation for the Energy related IQP and MQP's. The great questions asked, the feedback that was given, and new ideas where very helpful with designing and coming up with future ideas that could be added by a upcoming MQP team.

Table of Figures:

Abstract:

This project involved the design and implementation of a system to report to a homeowner his or her oil consumption in an easy to understand manner. A sensor and phone application were designed, built, and interfaced to track the run time of the furnace and process and report that data to the user on a graph. Energy consumption is a leading topic of discussion for homeowners and given the current economic climate, households are always looking for new and innovative ways to cut costs.

The sensors that has been designed, reports back to the application when the furnace turns ON or OFF. This data is then sent to an Android application we designed by means of Bluetooth LE where it is then processed and graphed. Bluetooth LE falls under the IEEE 802.15.3 Wireless Personal Area Network (WPAN) Standard. The graph of the data shows the daily oil usage of the furnace on the Y axis and the date on the X axis so homeowners can see different trends over time. The purpose of this graph style was so the homeowner could see his or her oil consumption trends and see how changes in the weather or temperature set on the thermostat affect the oil usage of his or her furnace.

Introduction:

The idea of coming up with ways to reduce one's oil consumption has always been a topic to save money. In recent years with oil prices rising and hitting record highs it has become a more popular subject. For the common homeowner in the Northeast, where they generally experience a colder climate, these rising oil prices have a very pronounced effect.

"The group said the national average cost to heat a home with oil this winter will be $2,593, up from $1,962 last winter. Families in cold-weather Northeast states will be hit even harder." According to an article published earlier this year from USA Today, you can see that the cost of heating a home with heating or fuel oil has gone up from last year.

Naturally homeowners are trying to get the best deal possible. With almost half of Massachusetts home owners using oil burning furnaces to heat their homes there was an opportunity to design some type of product that could possibly help homeowners reduce these costs.

In recent years we have seen a greater focus placed on energy usage reduction in the United States, and around the world. This can be seen in the recent successes of the Toyota Prius, a hybrid car, and Tesla Motors, a young company specializing in electric vehicles. Furthermore, we have seen great progress in renewable resources, specifically solar and wind technologies. The growing concern with this topic can be result of a few factors. The first is the rising price of oil worldwide. This has affected gasoline and oil prices and has helped to bring the idea of energy conservation to the forefront of people's minds. Here in Figure 1, we can see how the price of heating oil has also risen over the years of 2000 to 2010 compared to its closest competitor, natural gas. Heating oil today, in the Massachusetts area, is on average around $3.60 per gallon, whereas natural gas is around $2.42 for compressed natural gas. The second is the current economic situation, with the United States having gone through a recession that has affected people's livelihood. While the recession is now over, this has caused many people to try to save money whenever possible, energy bills included. The third reason is environmental preservation. In recent years the fear of climate change has led to an increased awareness of energy usage and the effects on the planet. While many homeowners understand these reasons to try and conserve energy, not as many have the tools to help them do so.

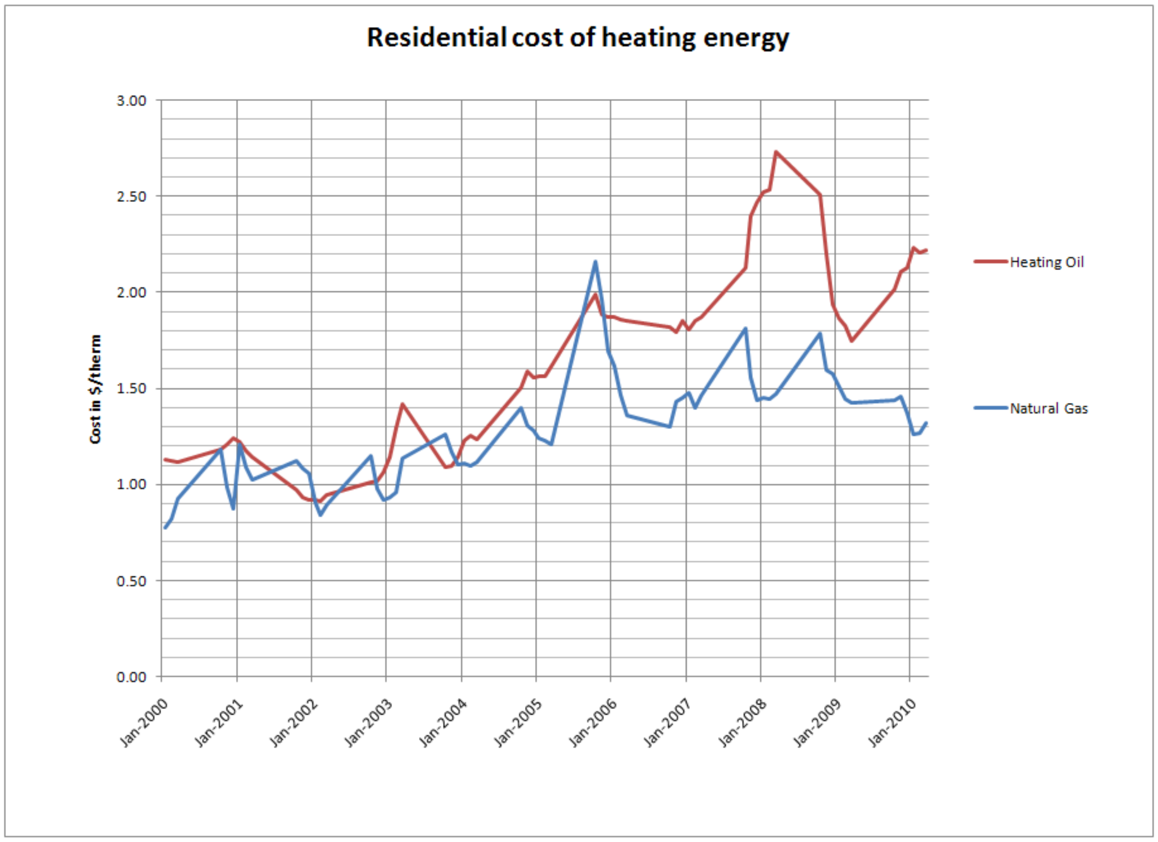


Figure #1: Oil vs. Gas Price

Below in Figure 2, you can see what the heating forecast for the year is going to be. Within the graph you can see how Massachusetts is compares to New England and the United States as a whole. The factors that went into the predictions for this graph were supply constraints that cause the price of fuels to rise, natural gas constraints, and winter electricity prices are expected to increase across all utilities. In Figure 3, the graph clearly depicts the average bill spent on heating ones home over the past six winters for different fuel types. The graph shows that heating oil is much more expensive than natural gas or electric heating systems.

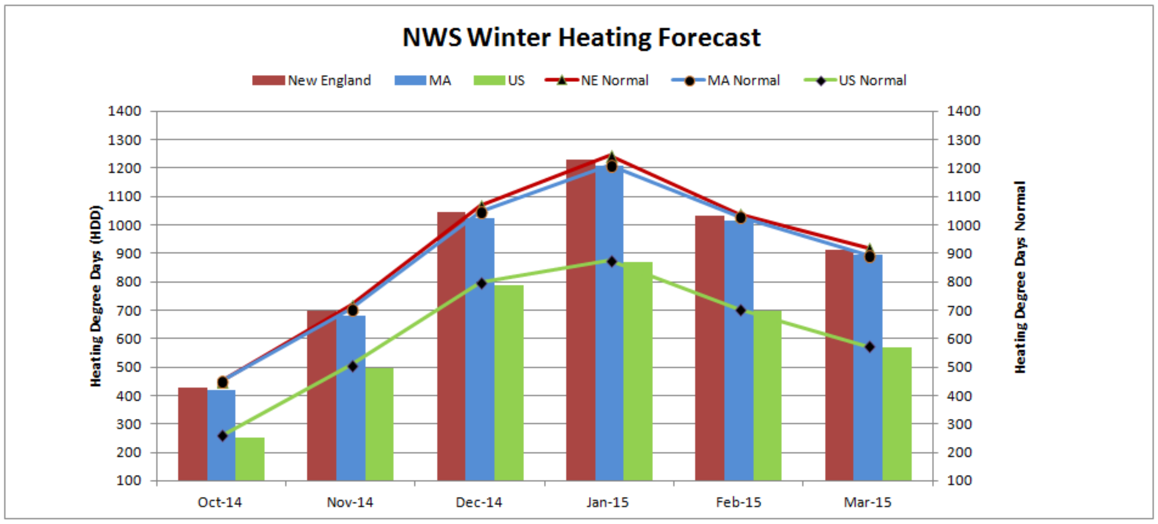


Figure #2: Winter Heating Forecast

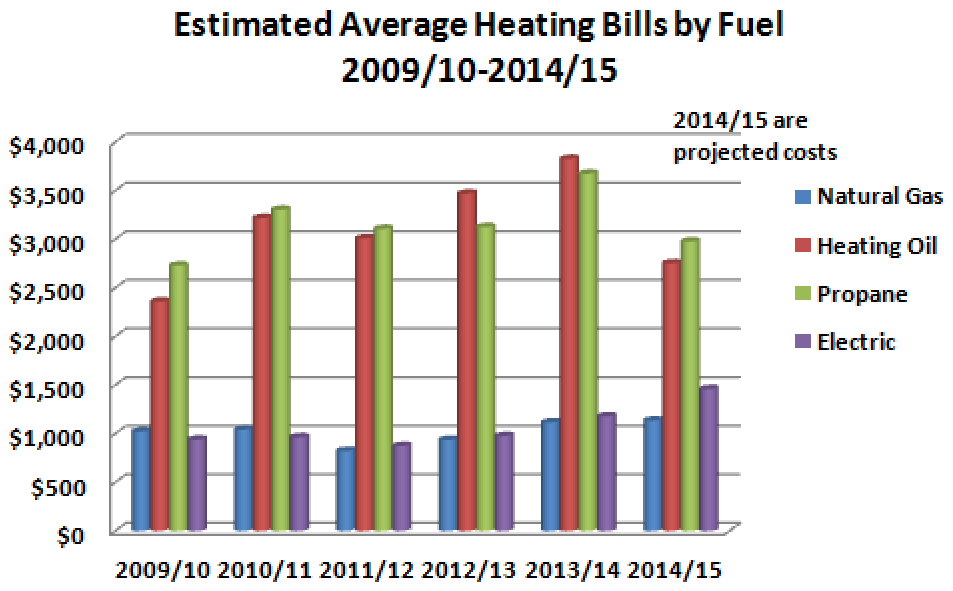


Figure #3: Average Heating Bills by Fuel Type

Electric and natural gas furnaces have the ability to be able to help homeowners track their energy use more easily thanks to meters outside the house, but oil furnaces do not have an equivalent. Homeowners with oil furnaces often rely on a small gauge on the oil tank, as well as a projected energy use given to them by the oil company to monitor the energy use of their furnace. Figure 4 displays an example of the gauge that imprecisely displays your oil level making it hard to track how much oil has been used. Oil furnaces are generally older technology that is not being phased out at fast as some would like. The availability of natural gas still is not available everywhere in the country, only being used by approximately half of the residential homes in the US.

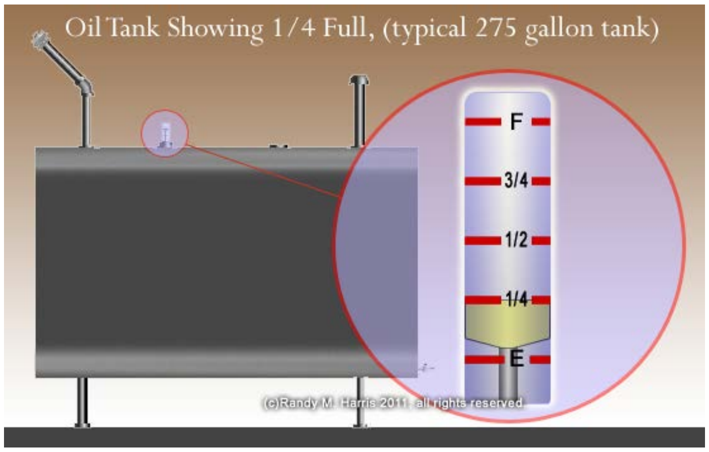

Oil Tank Showing 1/4 Full, (typical 275 gallon tank)
F
3/4
1/2
1/4
E
(c)Randy M. Harris 2011 all rights reserved

Figure #4: Oil Tank Gage.

Being the “old way”, oil furnaces leave room for these homeowners to miscalculate their energy use on a daily basis, which over time can amount to large discrepancies between their projected energy use, and the actual amount used. The goal behind this project is to produce a product for the average homeowner to monitor their oil consumption throughout a given period and use this information to reduce their carbon footprint on the environment and reduce financial and energy waste. The sensor should be able to monitor the oil usage of the furnace, and allow for this oil usage information to be presented to the homeowner in a clean and understandable format.

Background:

One of the fundamental aspects for this project was to learn how an oil furnace functions. As this project is aimed at sensing when an oil furnace is turned ON and OFF, we must understand in detail how a furnace works. Knowing the workings of an oil furnace should allow the team to choose the best way to sense the furnaces ON or OFF state.

1. Oil Furnace

One of the first and most important controls in a furnace is the thermostat. The thermostat is a very important control in a heating system because it turns the furnace on and off. There are a few different kinds of thermostats, but for the purposes of their integration into a heating system, they all work fairly similarly. They allow the user to set a desired temperature for the house, and once set, the thermostat will attempt to keep the temperature in the house at this temperature by switching the furnace on and off. The temperature range can be adjusted on some models to keep from the furnace short cycling, or turning on and off very often. For example, on some thermostats the user can adjust how many degrees the temperature of the room can differ from the desired temperature. The thermostat controls the furnace electronically with an on and off signal. This is most often done using a relay between the thermostat and the furnace itself, although it is theoretically possible to use a transistor, such as a MOSFET or BJT instead, allowing the thermostat, a relatively low power circuit to control the furnace, which, by comparison, uses much more power.

Once the furnace is given the signal to burn the oil from the thermostat, the pump on the furnace turns on and starts pumping oil from the tank through a spray nozzle and into the combustion chamber. Every spray nozzle is rated for a certain flow rate i.e. .85 gallons per hour. This constant flow rate for the spray nozzle means that the oil used by a furnace is linearly related to the time it is burning oil. This is a very important characteristic to keep in mind when developing a sensor to track oil usage of a furnace.

When the furnace is on, the pump forces oil through the spray nozzle into the combustion chamber. In this combustion chamber, a fire occurs which passes the heat through a heat exchanger to heat either air or water. In a forced air system, there is a fan or blower that blows hot air from the furnace throughout the house, while in a radiant heat system, there is a pump that circulates hot water through a series of radiators in the house. While these systems have some major differences in their heat delivery, for the purposes of this project they are similar enough to treat the same. The diagram below shows a forced air system.

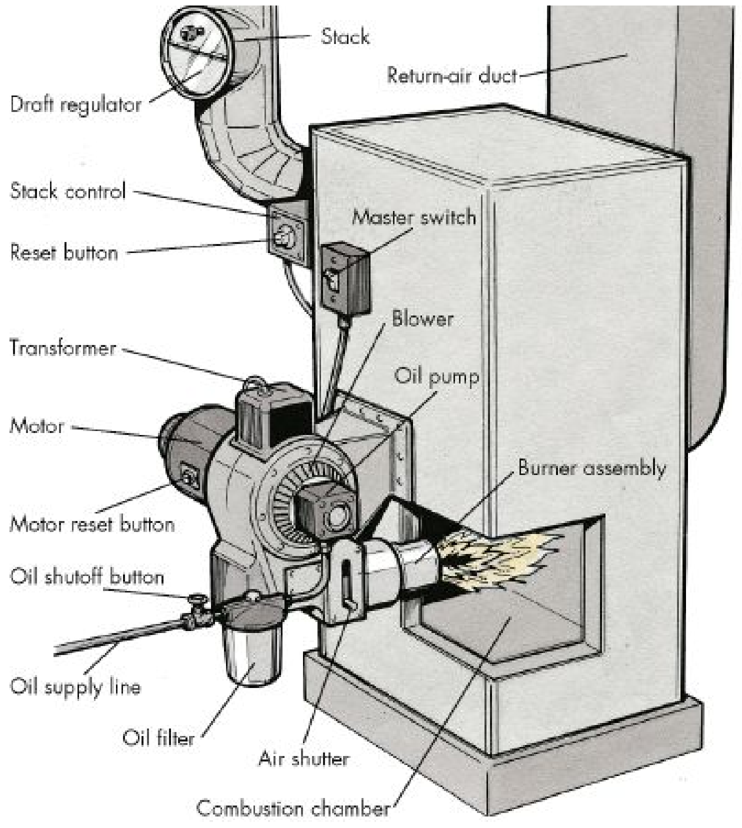


Figure #5: Oil Furnace

Apart from these parts of the furnace, there are also furnace controls, which are responsible for preventing a buildup of unburned atomized oil in the combustion chamber without igniting, and also responsible for limiting the temperature of the furnace to prevent heat damage. The prevention of this unburned atomized oil in the combustion chamber is handled by either the stack control on older furnaces, or on newer furnaces, an infrared flame sensor. Both of these detect the presence of a flame when the oil is being sprayed into the combustion chamber, and if no flame is present, will stop the flow of oil. This successfully prevents a buildup of unburned atomized oil in the combustion chamber, thus preventing a possible explosion. The second control is the limit switch. The limit switch has multiple purposes. The first is that it prevents the blower from turning on too quickly when the furnace first turns on, in order to prevent cold air being circulated throughout the home. The limit switch also has an upper limit, which turns the burner off if the temperature in the combustion chamber reaches the upper temperature limit. In this case, heat may still be being delivered to the house despite the furnace not burning any fuel.

There are multiple indications to an observer that the furnace is on. Two of the most obvious indications are that the furnace makes noise, and vibrates. This is mainly due to the aforementioned fuel pump and blower. Another observable change when most oil burners turn on is that the draft regulator opens up to some degree. The purpose of the draft regulator is to regulate the draft over the fire in the combustion chamber. If there is too much draft, there ends up being some heat waste, while if there is too little draft, the fuel may not burn completely thus wasting fuel, thus the draft regulator must be calibrated to each individual furnace. This ends up resulting in a different amount of measurable change in the draft regulator from furnace to furnace, with some homeowners reporting that their draft regulator hardly moves at all.

2. Power Source:

The power source that was needed for the choice of the system needed to be easily accessible for a "do it yourself" homeowner. The options were to use alkaline batteries, hardwire the sensor to the furnace, or plug the sensor into a wall outlet. With the thought of ease of installation in mind the best choice for power supply can be narrowed down to a wall outlet or alkaline batteries. Knowing that there is a possibility that the homeowner may not have an open outlet near the furnace, alkaline batteries become the best choice to keep from having a professional hardwire or install an outlet.

For safety reasons this is also the best choice. There are 120 AC voltages running thought a wall outlet in your home and unless you are a trained electrician this could be a very dangerous situation for someone trying to hardwire the sensor to the furnace. Using batteries may also have the benefit of drawing in more customers because of the ease of installation. With these factors it also made an easier decision to choose battery. One advantage to non-rechargeable batteries that was not identified is the ability for the user to use rechargeable batteries in the device instead. While this would end up increasing the cost to the user, it remains an option for those customers who would like to use rechargeable batteries.

2.1 Low Power Mode:

There are many different techniques that can be used to design a low power circuit. One of the most effective and simple ways to reduce power consumption within a circuit is to reduce the frequency of the digital circuits within the circuit. This helps to reduce the power consumption for two reasons. The first reason is that there is a certain amount of energy used every time a CMOS transistor transitions between the high and low states, thus if the frequency with which this occurs is reduced, less power is being consumed. The second reason is that the supply voltage of digital circuits using CMOS transistors can usually be lowered when the frequency is lowered. This process of setting the operating frequency of a digital circuit below the normal operating frequency is known colloquially as “underclocking” and can be an easy way to reduce the power consumption of a circuit if the normal performance of the digital circuits is greater than the performance required by the user.

Another simple way to reduce power is to have a sleep mode on the device. When a device can go into sleep mode it consumes much less power. In this application, this may be very plausible because there will only need to be transmission of the data when the furnace starts up and shuts off. This is a pretty basic technique, the device isn’t always running at full power, therefore it is not using as much power.

One alternative to creating a low power circuit is power harvesting. What this actually means is the circuit just draws power from somewhere else in the environment. The circuit can derive energy from external sources such as thermal energy, wind energy, salinity gradients, and kinetic energy. This captured energy can be stored for small devices. Energy scavenging can provide a very small amount of power for low-energy electronics. This is possible for our design by the choice of sensor that we selected in the piezoelectric sensor. The piezoelectric sensor creates a voltage as it vibrates back and forth. This is one way in the circuit that we might be able to harvest some energy. When the furnace turns on the sensor would vibrate creating the voltage. Since we believe that we only need to transmit data when it turns on this new voltage created from the vibrations could possibly be used to supplement the power used from transmission of the data.

3. Wireless Transmission:

3.1 Zigbee:

Zigbee is a fairly new short-range wireless network with a possibility for extremely low power consumption. ZigBee was conceived in 1998, it was then standardized in 2003, and revised in 2006 (IEEE 802.15.4 International Standard). The general range for a Zigbee personal wireless network is 10m to, roughly 20 meters. Zigbee is based on low-power digital radios. Although Zigbee has a low range transmission, it is capable of long rage transmission. It is able to do this through a different array of networking techniques outlined below.

This type of wireless network is usually used for low data transmission. The applications for Zigbee have a wide range from simple wireless applications such as light switches to industrial equipment with short range and low data transfer. This technology tends to be a simpler and more cost efficient than other wireless personal area network (WPAN) choices out on the market. Also it has lower power consumption for longer battery life making Zigbee a good choice for wireless control or monitoring applications.

The ZigBee network layer natively supports both star and tree networks. These are the type of techniques that can be used when the data may have to be transmitted over longer distances. Star networks are shaped like their name, a star.

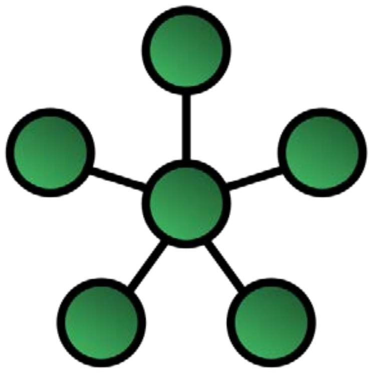


Figure #6: Example of Star Network

Star networks are one of the most common computer network technologies. In its simplest form, a star network consists of one central hub or computer, which acts as the main transmitter. This is basically the center point where all the other nodes connect; this central node provides the common connection point to all the other nodes. In star topology, every computer workstation is connected to a central node called a hub.

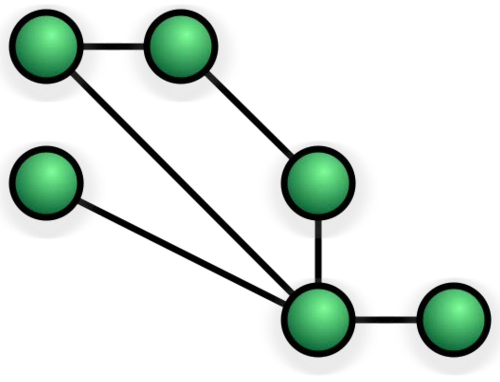


Figure #7: Example of a Tree Network

Tree topology is structured like a tree in the real world. The main idea behind the tree structure is it has a root node or the main node. All nodes off of this root node are intermediate nodes and can be called the branches and leaves. The root node is the main node of the structure, the branches are the nodes in between, and the leaves are the nodes that are last in line.

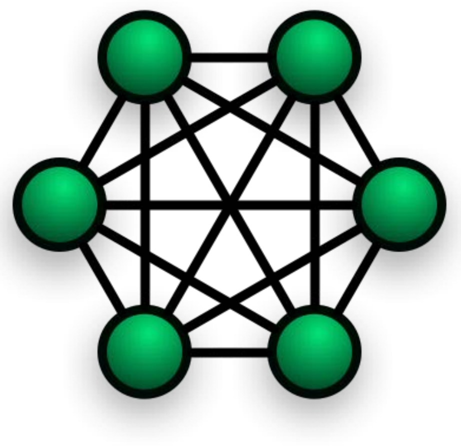


Figure #8: Example of a Mesh Network

A mesh network can be designed using a flooding technique or a rerouting technique. When using a routing technique, the message is propagated along a path, by hopping from node to node until the destination is reached. A mesh network whose nodes are all connected to each other is known as a fully connected network.

For our situation with using ZigBee form of transmission would be simple with very few nodes. The most logical form would be the Star Network with our sensor device being the central hub connecting and transmitting its info to two to three devices such as cell phones or computers. ZigBee is used for much longer range than really necessary for the idea at hand. One benefit ZigBee technology has is that it has the capability of being lower power than Bluetooth or Wi-Fi networks.

3.2 Wi-Fi:

Wi-Fi is a local area wireless network technology. A Wi-Fi network allows the transfer data to the Internet or other devices on the bands from 2.4 GHz to 5 GHz radio waves (which falls under the IEEE 802.11 radio standards). To connect to a Wi-Fi Local Area Network, a device needs to be equipped with a wireless network interface controller. The combination of device and interface controller is called a station that shares the data through a single radio frequency communication channel. All stations within range receive transmissions on this channel. The hardware does not signal the user that the transmission was delivered. A carrier wave is used to transmit the data in packets, referred to as Ethernet Frames. Each station is constantly tuned in on the radio frequency communication channel to pick up available transmissions.

Wi-Fi networks have limited range. A typical wireless access point with a stock antenna might have a range of 35 m indoors and 100 m outdoors. Due to reach requirements for wireless local area networks (LANs) applications, Wi-Fi has fairly high power consumption compared to some other standards. Technologies such as Bluetooth provide a much shorter range between 1m to 60m and so in general have lower power consumption. Other low-power technologies such as ZigBee have a possibility of a fairly long range when connecting networks, but much lower data rate. The high power consumption of Wi-Fi makes battery life in mobile devices a concern. See figure # 6 for a comparison of the differences between the wireless networks.

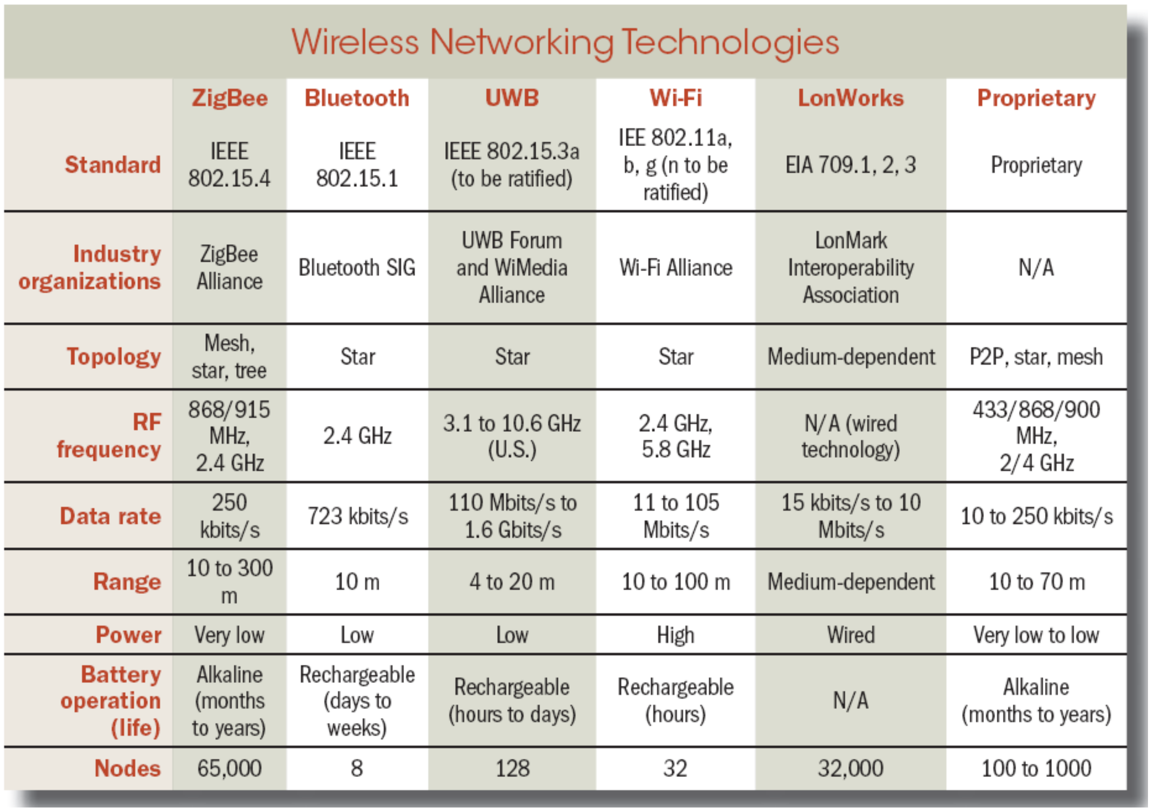
Using a Wi-Fi connection to transfer the data from the sensor may be the best option in the event of posting the information on a web page. Since Wi-Fi tends to deal with the Internet this would be the easiest way to transfer data. In this case if it is decided that the information should be posted to a web page it will be able to be accessed by any device with the Internet capability. The downfall to this is when trying to keep power consumption to a minimum will be difficult because Wi-Fi has the highest power consumption.

3.3 Bluetooth:

Bluetooth technology has made its strides in the tech market for short-range wireless data transfer. Bluetooth is a wireless technology standard for exchanging data over short distances using short-wavelength radio wave frequencies from 2.4 to 2.485 GHz (IEEE 802.15.1 International Standard). The signal frequencies come from fixed or mobile devices to create a Personal Area Networks. Bluetooth is capable of a transmission range of 1m to roughly 10m. This is in the globally unlicensed but not unregulated Industrial, Scientific and Medical 2.4 GHz short-range radio frequency band.

A master Bluetooth device can communicate with a maximum of seven devices. The network topology standard for a standard Bluetooth device is a star type network. The number of device connected should not be of concern with the situation at hand, although seven devices are able to connect, our design should only be connecting to one or two devices most likely being connected to the oil burning furnace detection sensor. The devices can be switched between the role of master and the slave.   At any given time, data can be transferred between the master and one of the other devices. The master chooses which slave device to address; typically, it switches rapidly from one device to another in a round-robin fashion.

The master is the one that chooses which slave to address and send data. While the slave has the role of listening and receive the data being transmitted from the master unit. The master has a slightly lighter load than the slave. Being a master of seven slaves is possible; being a slave of more than one master is difficult but would have no relevance in the task of the oil burning furnace detection sensor.



Wireless Networking Technologies

	ZigBee	Bluetooth	UWB	Wi-Fi	LonWorks	Proprietary
Standard	IEEE 802.15.4	IEEE 802.15.1	IEEE 802.15.3a (to be ratified)	IEE 802.11a, b, g (n to be ratified)	EIA 709.1, 2, 3	Proprietary
Industry organizations	ZigBee Alliance	Bluetooth SIG	UWB Forum and WiMedia Alliance	Wi-Fi Alliance	LonMark Interoperability Association	N/A
Topology	Mesh, star, tree	Star	Star	Star	Medium-dependent	P2P, star, mesh
RF frequency	868/915 MHz, 2.4 GHz	2.4 GHz	3.1 to 10.6 GHz (U.S.)	2.4 GHz, 5.8 GHz	N/A (wired technology)	433/868/900 MHz, 2/4 GHz
Data rate	250 kbits/s	723 kbits/s	110 Mbits/s to 1.6 Gbits/s	11 to 105 Mbits/s	15 kbits/s to 10 Mbits/s	10 to 250 kbits/s
Range	10 to 300 m	10 m	4 to 20 m	10 to 100 m	Medium-dependent	10 to 70 m
Power	Very low	Low	Low	High	Wired	Very low to low
Battery operation (life)	Alkaline (months to years)	Rechargeable (days to weeks)	Rechargeable (hours to days)	Rechargeable (hours)	N/A	Alkaline (months to years)
Nodes	65,000	8	128	32	32,000	100 to 1000

Figure #9: Wireless Network Technologies

4. Sensors:

4.1 Piezoelectric Vibration Sensor:

When it comes to sensing the state of the furnace there are a variety of techniques and sensors available to use. As mentioned previously, when an oil furnace is on, it creates a significant amount of vibration. This is one possible area to design a device around. One of the most common vibration sensors is the piezoelectric vibration sensor. This sensor works by using an unconstrained mass on one end, and a crystal capable of producing a piezoelectric effect, such as quartz, on the other end. The piezoelectric effect refers to an effect observed when pressure is applied to certain special materials. These materials transform the mechanical energy of pressure into electrical energy. The sensor in this case, works by allowing the mass to pivot in one dimension, and as this mass pivots pressure is applied to the crystal, which generates electricity. In the case of a vibration, the mass would repeatedly move back and forth, causing the crystal to give an oscillating output. The sensitivities of these sensors are typically measured in volts per unit of acceleration due to gravity, or V/g, with the sensitivity being highest at the resonant frequency of the sensor. Depending on the mass, these sensors can be configured to have resonant frequencies as low as low as 20 Hz or upwards of 100Hz. This sensor would most likely require tuning the mass in order to get the strongest signal at the resonant frequency of the furnace.

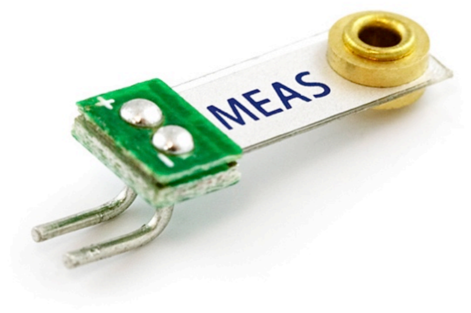


Figure #10: Piezoelectric Vibration Sensor

3.2 Microphone:

In addition to the vibration generated by a furnace, noise is also generated. This could also provide a simple method for creating a product to sense the state of the furnace. This would require the use of a microphone. While there are many types of microphones, dynamic, condenser, and ribbon being just a few, the type with the most relevance to this project is the electret microphone. Electret microphones are relatively small and cheap to produce, with one of their major uses being in cell phones. The sensitivities of these devices are often measure in volts per Pascal, or V/Pa, and they can have frequency responses as wide as 10Hz to 30kHz, wider than the human range of hearing. Some drawbacks to these microphones are that they can have high noise floors, high distortion, and uneven frequency response. While the last 2 drawbacks do not really concern this project, the first will have to be taken into account if an electret microphone is used, though the noise floor does vary from microphone to microphone.

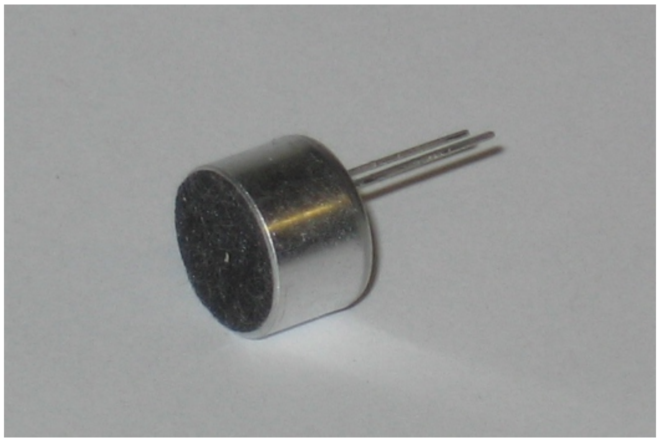


Figure #11: Electret microphone

4.2Electricity Sensing:

The third and final sensing technique to be discussed is a purely electrical sensing technique based around the thermostat. This technique would most likely not work as intended though, because while the signal from the thermostat may be telling the furnace to stay on, the stack control, or limit switch on the furnace could turn the furnace off despite the signal from the thermostat. On the other hand, this technique wouldn't require a "sensor" in the traditional sense, as our device would simply connect electrically to the thermostat. One other benefit to this technique is that it might be possible to draw power directly from the thermostat.

5. Processor:

One of the most important factors when making a low power system is to design something with as few components as possible. With this in mind, it was determined that the sensor system needed a microprocessor, because without it, the functionality would be severely compromised. For example, the software side of the project necessitated that the sensor send out a timestamp when the state of the furnace changed. This would not have been done as easily without the use of a microprocessor and most likely would have ended up using up more power and space if done purely in the analog domain. Another important factor in the design was that users be able to use their smartphone normally, meaning the smartphone would not always be within range to the sensor. This necessitated that the usage data be kept on memory in the sensor until the smartphone came back into the range of the sensor, which once again could be implemented much more easily digitally.

For our choice of microprocessor, we went with the MSP430 family of microprocessors. There were two main reasons for this choice. The first reason was familiarity. Both students working on this project have experience programming this family of processor, which should allow for quick development of the software as well as a more reliable product at the end of prototyping. The second reason for choosing this family of microprocessors was the power draw. In a datasheet provided by TI, a thermostat built using the MSP430 microcontroller used was found to have a battery life of over 15 years on a single AA battery. While our application will most likely use more power, this is encouraging because that battery life is an order of magnitude larger than the desired runtime of 1 year.

Product Goals:

The main goal of our oil detection sensor was to design a functional, cheaper alternative to compete with expensive thermostats as a way to track and try to reduce the amount of oil that is being burned in older less efficient oil burning furnaces. As the team’s sensor is not an alternative to a thermostat, it is an addition or another option for a homeowner that allows them to track how much heat fuel has been burned in a given time frame. With a product like this it would be a helpful tool for any home and smart phone owner to try to reduce the amount of money that they are spending to heat their home by just being smarter about monitoring when to run their furnace.

For our product to be successful there are some major requirements that must be fulfilled. While all of these requirements may not be able to be fully fulfilled simultaneously, the goal is to balance all of them to create the most enticing product for the consumer. The following is a list of our product requirements:

- Easy to use
- Affordable
- Durable
- Long Battery Life
- Low power
- Accurate

The first product requirement is that it is easy to use. Due to the fact that the target market for this product is the average homeowner, it is important to deliver a sensor system to them that can be easily installed and used. Making it as simple as possible to understand the product design is so important. This would fully extend the information that is presented to them. The user interface of the app must be simple too while being able to get the information they desire quickly and easily.

The second product requirement is that it is affordable. This is a completely relative term depending on the market, but for this product in a market where the average cost of heating a home for the winter in the Northeast region of the United States is expected to jump to over $2500, the target MSRP is to be under $75 with the application included. Relative to the price of heating their homes, this would represent a small amount of money, and allow customers to save money in the future.

The third product requirement is durability and longevity. Ideally, the sensor system will last 5 years, with 3 years being the absolute minimum. This is important because the combination of the price and the lifespan of the device can impact the perception of value to the customer. No matter how cheap the device is, if it has a short lifespan, the value to the consumer is immediately reduced. On the other hand, no matter how durable and long lasting the device is, if it is priced too high, then the value is also reduced.

The fourth requirement is low power. In an ideal world, the sensor would be able to scavenge some power from the surrounding area, but for this project, it seems like battery power is inevitable. With battery power, the goal is a battery lifespan of at least one year for the prototype, hopefully longer than that though.

The final product requirement is accuracy. This is very important because if the product is not accurate, it quickly becomes useless. At this point we hope to be 98% accurate on the timescale of one month. That is, the fuel usage we claim to have detected should be within 2% of the actual fuel used by the furnace.

1. Market Competitor:

There are multiple competitors on the market in the form of smart thermostats. One that is the most similar, in terms of functionality to the teams Oil Detection Sensor is the Nest Thermostat. As the main competitor to the Oil Detection Sensor the Nest product is a $250 smart thermostat depicted below in Figure 9. The Nest has the ability to connect to a phone, set a schedule, track your energy use, as well as set temperature wirelessly from wherever you are. This is different than the teams Oil Detection Sensor because the sensor is not a thermostat and does not have the ability to charge the temperature setting. The sensor is designed to track the furnaces run time and display the oil usage for the home owner to decide when they want the furnace running.

The major differences with our product is the fact that you would have to manually adjust the thermostat since our product is not attached to the thermostat at all, but attached to the furnace itself. Where it does have the possible benefit of being much cheaper would be through its simplicity and zero cost for installation. With the competitor, Nest, it is recommended that your heating and cooling professional install it, which would cost more because of the installation bill. Our product is installed by simply attaching the housing of the sensor to the housing of the furnace with the use of magnets that will be provided.

Figure #12: Nest Thermostat

Our product has the ability to connect with the homeowner's smart phone by using the Bluetooth certification standard (IEEE 802.15.1), which falls under the Wireless PAN standard (IEEE 802.15). This allows the homeowner to access the data whenever they please and whenever they are within range of the unit. This can be looked at as a benefit or hassle. It would be beneficial because the application is not always running along with always wasting battery trying to receive new data throughout the day.

2. Connection:

As previously stated the smart phones connected to the sensor through Bluetooth Low Energy (LE) and should be able to collect the new data that has been stored on the MSP430 chip on the sensor whenever the homeowner pleases. Since Bluetooth devices are only PAN networks, that means when the smart phone is not connected to the sensor, or central hub, and you exceed the range of the Bluetooth antenna, the data that is collected while away needs to be stored and transmitted at a later time without losing data points. Once collected the data should be graphed and oil consumption calculated on the phone application for the homeowner.

Bluetooth LE was built with the idea that the slaves may or may not be connected to the center at any time. Thus, the network topology of Bluetooth LE lends itself to this application. The smartphone in this case is the hub of the network, connecting and disconnecting to the sensor at will, and when connected polling the sensor for the data. This allows for the user to continue using his or her smartphone normally.

3. Low Power and Longevity:

Bluetooth LE is commonly used within the medical and sports and fitness accessories. It is more beneficial in our case also because of the lower peak current draw than normal Bluetooth connection. Low Energy has the benefit of being able to run for years on batteries, as well as lower implementation expenses, and low idle current draws.

The main goal behind having a system use the lowest energy possible was to achieve the longest battery life possible. With making the system as simple as possible the team wanted to be able to use simple AA or AAA batteries for ease of use for the consumer that would potentially be buying the product. The team ended up choosing 3 AA batteries

Methodology:

In order to design the sensor and the system an assortment of parts will need to be chosen. The design of our system will have multiple options in the type of sensor, the different wireless technologies to transmit the information, the type of battery that will be the best fit for our requirements, and finally the microcontroller. In order to choose these parts a value analysis will be performed. In the following section our analysis was done to show which parts may be most suitable for our application in our sensor and system design.

1. Original Design:

In figure 10 below, a flow chart of the design process that the team took for the approach on the design of the sensor is depicted. This flow chart is what the team had originally used during the start of the design phase. It was decided that the first step was to somehow detect the state of the furnace. In this case either the furnace is completely ON or completely OFF. This made this part easy because of only two different state changes. Next was figuring out how to store this data locally on the sensor itself. For this the team used the MSP430 microprocessor to log the time of these state changes and store them to the MSP's flash memory. This data then has to be sent from the sensor to a smart phone. Once the data is received at the smart phone the data needs to then be processed and displayed in a way that is helpful for the homeowner. In this case the team went with a graph that can easily display the fuel that is being burned and when that fuel was burned.

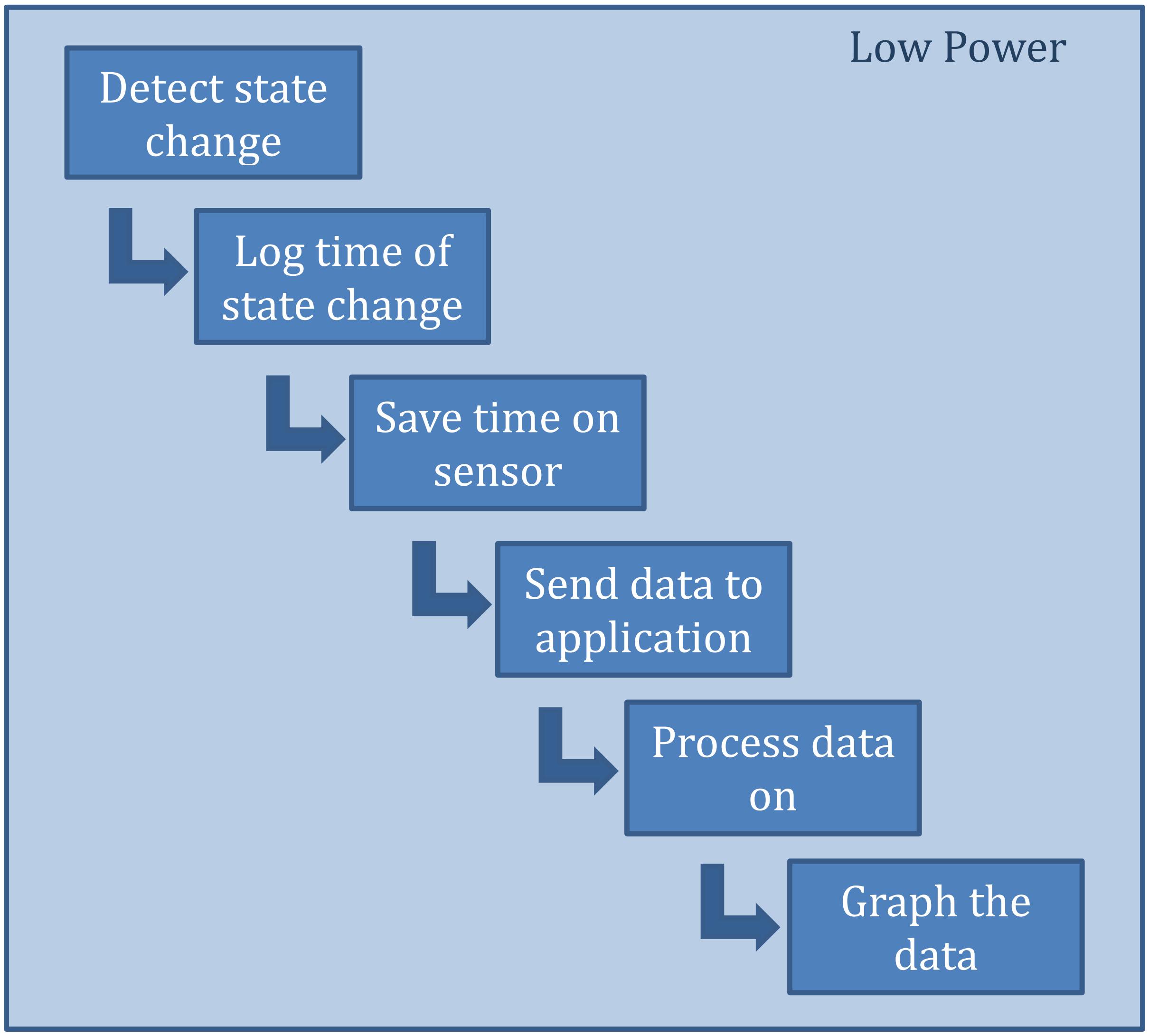


Figure #13: Design Process flow Chart.

In figure 11 below, the high-level block diagram shows the process of how the team went about designing the sensor. This is only the block diagram for the hardware circuit. Looking closer at the block diagram it shows how the battery is feeding the buck converter voltage regulator. From there the voltage regulator is taking the 4.5 volts from the three AA batteries and regulating it down to the desired 3.3 volts that the other components need to operate. The Piezoelectric sensor detects the state change from the furnace. There are only two different states in this case, either the furnace is ON or OFF. From there the signal needed to be amplified before being sent and stored in the microprocessor. The signal from the piezoelectric sensor should be amplified so the data is more easily tracked and saved to the microprocessor.

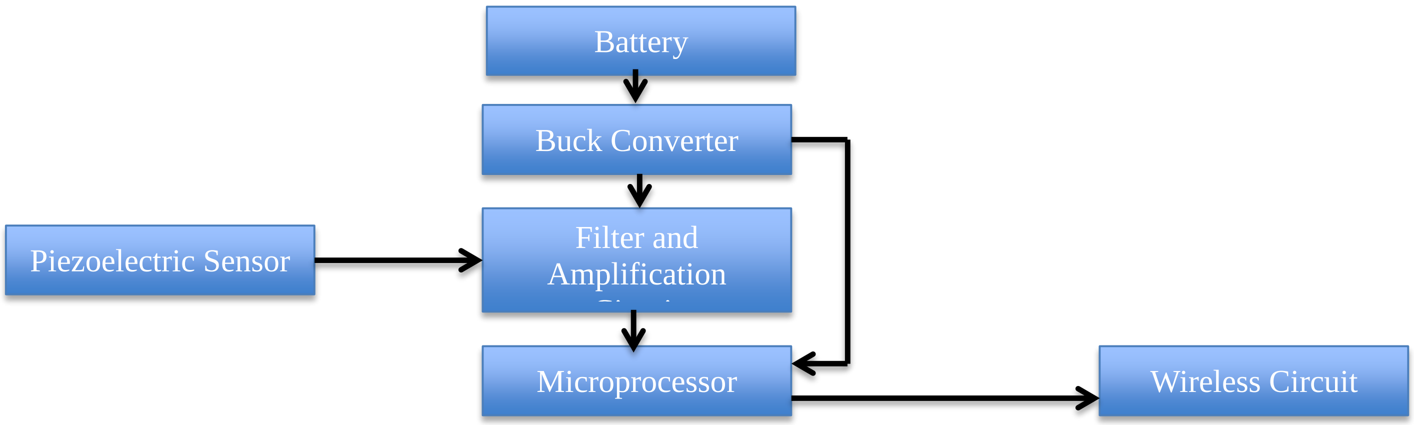


Figure #14: Hardware Block Diagram

2. Changes to Original design:

Once the original design was created minor changes had to be made to achieve the performance deemed necessary in our proposal. These minor changes were simple part swaps due to power requirements not being met. Both the OP amp and buck converter needed to be changed out due to their relatively high quiescent current. These crucial parts of the system can be seen outlined in a schematic that has been added at the end in Appendix B.

The new buck converter and OP amp also came with the benefits that they were more readily available. These parts were about the same cost as the parts they replaced as well, so the change had no negative impact on our product requirements.

The resistor values for the OP amp circuit were also changed in order to create a more suitable gain of 67 for our circuit. This value was chosen because the measured output voltage of the piezoelectric sensor was roughly 75mV while the ADC on the MSP430 goes from 0 to 3.3V. With a gain of 65-70, this comes out to roughly 3.3V at the output, which works great for our purpose. The schematic changes can be seen added in at the end in Appendix B as the last schematic. Some oscilloscope images from early stages of test that depict the gain of the system can been seen later on in the testing section.

One of the few drawbacks of the current design is that the Bluetooth module that is in use is only readily available from China. While we were able to locate one in the United States on eBay, if another is required on short notice it will most likely create another delay in the development cycle.

3. Bluetooth Connection:

Originally the connection of the Bluetooth module to the smartphone appeared easy, but unfortunately this turned out not to be the case. Finding a suitable Bluetooth LE module for our application was fairly simple, but after that the process became relatively complicated. Interfacing the module with the MSP430 over Bluetooth was more difficult than we had initially anticipated, but by far the hardest part of the Bluetooth connection was connecting the application and the module and sending and receiving the data between the phone and the sensor reliably.

The application development was done in the official Android IDE, Android Studio in JAVA. Coding the application was very difficult because of the limited experience we had with JAVA. While other pieces of the application such as the graph were fairly simple, the Bluetooth side was quite difficult. Initially everything was done using the older Bluetooth APIs but upon trying this, we realized that we needed to use the Bluetooth LE API which is much more difficult to use as it uses GATT profiles and other systems that make serial communication much more difficult. We ended up using another application that we found the source code for which was a serial communication app, and modifying it to include the graphing code. This proved fruitful and luckily we got the app to work properly.

There is an app that came included with the Bluetooth module that made debugging very simple. Initially, we simply established a connection to the module. After the connection was confirmed, we moved on to interfacing the module with the MSP430 over UART (EIA standard). Once we programmed the MSP430 we were able to send and receive commands. This was initially verified by turning an LED connected to the MSP430 on and off using a command sent from the app.

4. PCB Design:

To design the PCB for the first prototype of the project it was decided that the company Advanced Circuits was the best fit for a simple board that the team needed. When designing the board it was fairly straightforward by using MultiSim and UltiBoard, two National Instruments programs that are accessible on the WPI campus. MultiSim is a very powerful schematic capture and simulation environment for not only students, but professionals too. UlitBoard is a similar product that goes hand in hand with MultiSim. What UltiBoard is, is a PCB design environment program that helps accelerate the design of PCB's with automated functionality while maintaining precision.

The team used MultiSim to draw up a schematic with all the selected parts. MultiSim has a wide variety of parts with different layout packages, but unfortunately it did not have some of the chosen parts for the project build. This made the schematic drawing and PCB design only slightly more difficult because it required the design of the part specifically within the program itself. This provided a minor step back in the finalization of a design within UltiBoard but was quickly overcome with the help of Joe St. Germain.

Once the schematic was finalized on the MultiSim it was then transferred over to UltiBoard. Within UltiBoard the team went about designing the layout for the board, which went smoothly after the tutorial from Joe St. Germain. With the board being fairly basic, only a two-layered PCB was needed as a finished product. This was also a benefit to keep the cost of the final product down as well. Below in Figure 15 is the final PCB when it has been complete and populated with all the components.

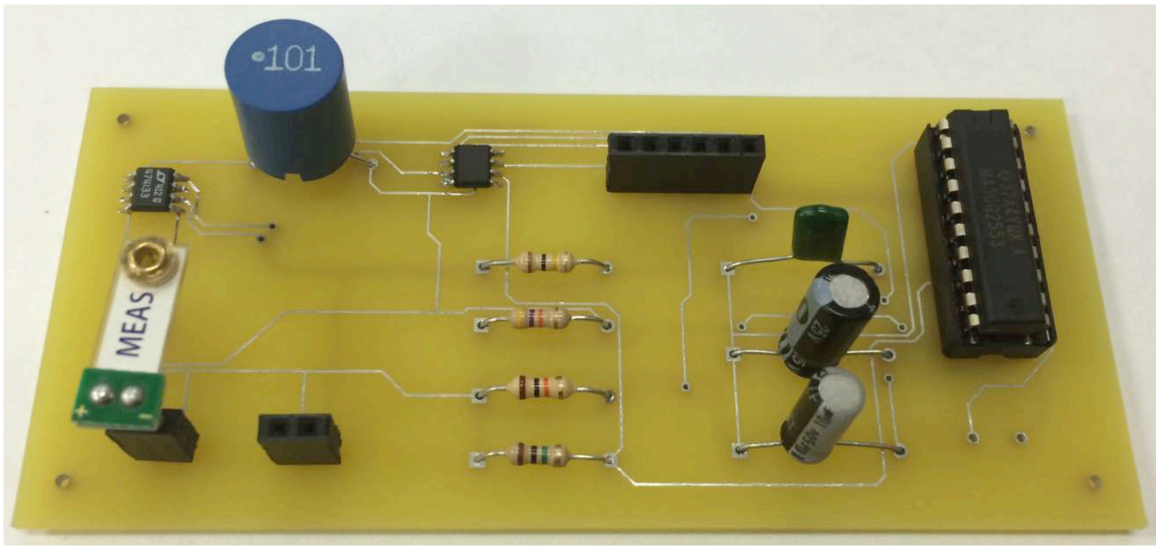


Figure #15: Populated Final PCB

5. Android Application:

The development of the application is only Android compatible at the present moment. The application can communicate with the sensor wirelessly and display the graphically as a visual aid to see when the furnace was running and for how long. The graph is a line graph showing the oil usage per day, in gallons. The user can adjust the timescale for the graph as well.

For developing an Android application we used a program called Android Studio, version 1.0.2 for Mac OS X. This is an integrated development environment or IDE and all the programming had to be done in Java. The team originally lacked experience with Java, but now is more proficient because of the gain of experience through self-taught trial and error throughout the design of the project.

The block diagram below shows the function of the application at a fairly high level. The application retrieves the raw data from the sensor. This will be done over Bluetooth LE connection to the phone. The raw data is time stamped to display of when the furnace turned on and off. This data will allow us to calculate how long the furnace was on for and from there can display the data as a graph on the smartphone. This graph shows the homeowner their energy usage over time allowing the user to see how much oil they have accurately burned over the given time scale.

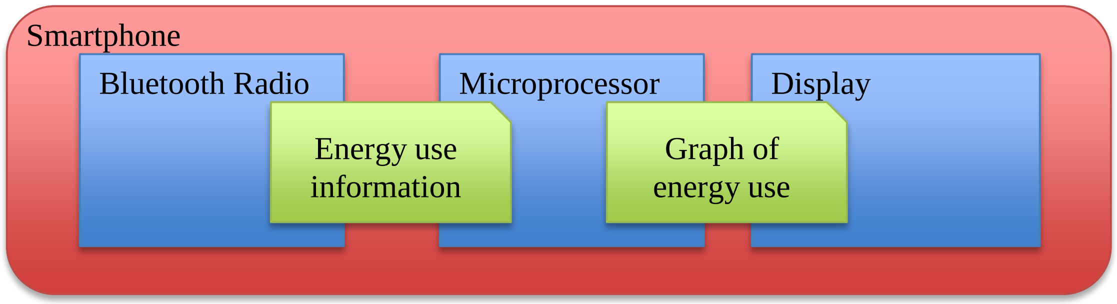


Figure #16: Smartphone software block diagram

Another important factor in the software is the calibration of the sensor. As was documented in the background, furnaces consume oil at a fixed rate when generating heat, but this rate differs between furnaces because the nozzles differ in ratings. Often times this rating is not known to homeowners so obtaining this information will need to be done in a different manner. One of the simplest solutions to this problem is to have a "calibration period" for the sensor. During this calibration period, the user would fill the oil tank for the furnace to a known quantity, and the sensor would record the total duration that the furnace was on while that known quantity was fully consumed. Then, by having both the duration that the furnace was on and the quantity, the application could easily calculate the rate at which the furnace burns oil by dividing the total quantity burned by the duration that the furnace was on. While this approach is simple and effective, it does have one major drawback and that is that it is fairly time consuming because the furnace will have to burn through a considerable amount of fuel.

One of the major hurdles when implementing the application was establishing a Bluetooth connection. While the documentation for implementing Bluetooth into an android application is abundant, the same is not true for Bluetooth LE. Thus our approach was to find working source code for an existing application and modify it so that the application could work for our needs. These modification mainly revolved around changing the user interface to include a graph, and then adding in the necessary code behind the scenes to process the data from the sensor and display it on the graph.

6. Case:

The case for the sensor is a simple acrylic box that encloses the hardware. The case was designed on the SolidWorks software program on the WPI campus. Once the design was finalized on the computer the team was then able to bring it over to the

Washburn workshop and use the laser cutting to cut out the 6 sides of the enclose. There is a small cut out on the top of the case where the Bluetooth module is able to stick out slightly to ensure that there is less obstruction for the antenna. This first prototype of the case is a little large for a first design. It does have the room to be made smaller but the team had decided against stacking the battery and PCB because size was not a real constraint.

In the Design the team discussed a few options and possible problems. With choosing to attach the sensor and enclose to the furnace via magnets, the first choice was using magnets, but a worry of the magnetic field disrupting any of the chips on the PDC board. The other choice for attaching the sensor to the furnace was Velcro. But the problem of the Velcro absorbing some of the vibrations need for data collection through the piezoelectric sensor. The final choice was the magnets because the fields in theory should not disrupt anything.

7. User and Installation Guide:

Since this is product that has the possibility of being sold to the public there is a user manual and a do it your self-installation guide that has been attached in Appendix E. This is a quick tip guide for the customer to be able to solve any of the questions that they might have after purchasing the product. It has a quick layout of how to install the application on a smart phone as well as install the sensor to your furnace at home. It also includes a small trouble shooting section were some problems that were encountered during the design and build of the sensor have been outlined with a solution of how to fix them.

8. Sensor code build:

With the circuit fully designed and created on the proto-board we could start moving forward on the software for the sensor. The plan was to break the code into chunks instead of trying to write all the code at once. This way when problems arose, finding the problem and debugging it would be simpler. Approaching the code like this also has the benefit of having functional milestones to present to others to demonstrate progress.

The first goal for the piece-by-piece build was to have the processor and proto board preform a simple action such as having the board preform an ON or OFF function. This would be done by turning ON and OFF a few LED lights. This was done by sending an "R" or "r" and "G" or "g" to the Bluetooth module, which was now added to our circuit, to turn on or off the Red and Green LED. Upper case letters turned the LED on while lower case letters would turn the LED off.

The next goal was getting a timer to work. This is crucial because the sensor must know exactly how much time occurred between furnace cycles. To confirm that the timer was running correctly the green LED was changed to alternate turning on and off every second. Along with the visual flashing of the green LED, when the letter "T" was sent wireless from the phone to the MSP430, the MSP430 would return the number of seconds that had passed. Initially the timer was only counting 10 seconds at a time, but eventually it was expanded to count minutes and hours as well. The only problem at this point was that the time could not be set.

The original idea on how to set the time was to allow the phone to send the current time to the MSP430 and have the MSP430 keep time. We did accomplish this and if the "s" character was sent the included app would prompt the user to enter the hour, minute and second on separate prompt lines, and the MSP430 would keep the real time. This idea was quickly dismissed once it was time to implement that flash though, as the amount of memory needed to keep the exact time was too large. Instead it was decided that the MSP430 would simply count the number of 30 second cycles that has occurred since the last synchronization with the app.

The ADC code was one of the most time consuming blocks of code that we encountered during the programming process. The first attempt made to help this process was to look at example code from ECE 2049, but while the microprocessor used in that class was also an MSP430, the control registers were different. This meant that more example code from other sources on the Internet was needed. Eventually, the ADC was implemented, but only after first implementing it in its own project, and then moving the necessary code over.

A very important part of the build is the flash code. The flash is where the data will be stored upon capture before sending to the app when it is synced with the sensor. As of now the flash works. Data can be written to and read from the flash. One of the difficult parts of implementing the flash stemmed from the fact that data in flash cannot be overwritten; it must first be deleted first. This, combined with the fact that the flash must be erased in 512 byte segments at a time, means that overwriting small amounts of flash is something that should be avoided at all costs. The current flash implementation only overwrites the flash once the data has been transferred to the phone. This technique avoids repeatedly expending energy to write and erase the flash.

9. Code Simplification:

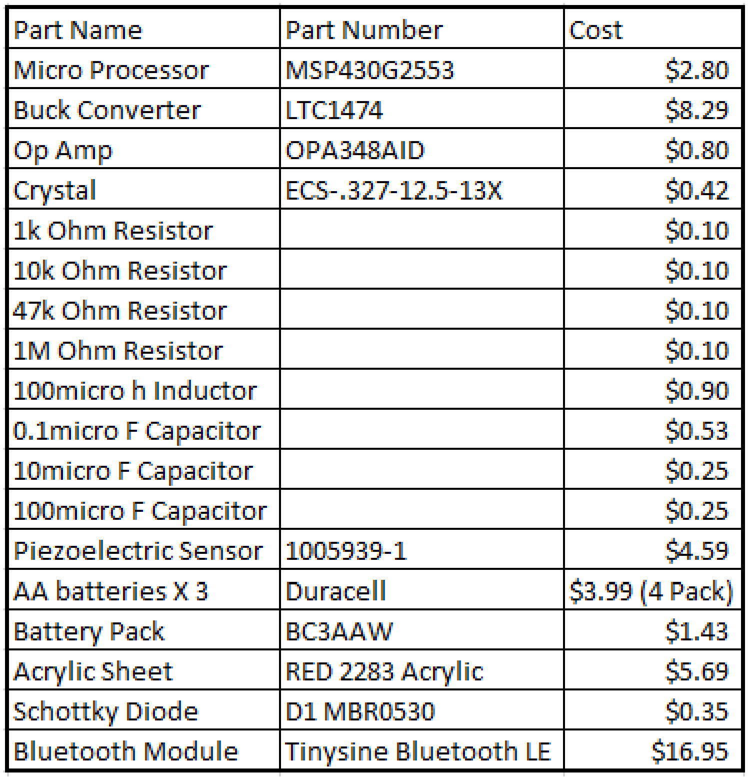
A plus to using a smart phone is that it has a much more powerful microprocessor than the MSP430 and thus can be used to do more of the complex tasks. Because of this this, we have thought some of our tasks that we are having the MSP430 do now, may actually be better to have done on the phone. One change is to keep the absolute time on the application and keep the relative time on the application. By this we mean that the application would keep the date, while the microprocessor will just count the number of 30-second ticks that have occurred. Once the data is received from the phone the application will update the time and the number of ticks will be reset to 0 so the sensor will be measuring from the updated time.

Cost and Budget:

For this project the team went in with the one of the goals to keep costs as low as we could. With a fairly simple design, a good portion of the work being done on the homeowner's smartphone, and a free app development program this proved to be a fairly easy task.

One point to make on how to keep cost as low as possible when thinking of mass production is to look at the cost of the parts on a mass scale. During our prototyping phase we may end up paying slightly more per part simple because the quantity that we are buying these parts in are. The cheaper the parts used in our prototype, the cheaper the finished final product to the consumer would be therefore.

Below in Figure #17 is a list of parts used with the project and their costs. The part numbers of the capacitors and resistors were left out because they are not specific and are interchangeable with other capacitors and resistors of the same value.



Part Name	Part Number	Cost
Micro Processor	MSP430G2553	$2.80
Buck Converter	LTC1474	$8.29
Op Amp	OPA348AID	$0.80
Crystal	ECS-.327-12.5-13X	$0.42
1k Ohm Resistor		$0.10
10k Ohm Resistor		$0.10
47k Ohm Resistor		$0.10
1M Ohm Resistor		$0.10
100micro h Inductor		$0.90
0.1micro F Capacitor		$0.53
10micro F Capacitor		$0.25
100micro F Capacitor		$0.25
Piezoelectric Sensor	1005939-1	$4.59
AA batteries X 3	Duracell	$3.99 (4 Pack)
Battery Pack	BC3AAW	$1.43
Acrylic Sheet	RED 2283 Acrylic	$5.69
Schottky Diode	D1 MBR0530	$0.35
Bluetooth Module	Tinysine Bluetooth LE	$16.95

Figure #17: Parts List and Costs

Testing:

1. Discrete Components:

As far as testing goes for the development of the sensor system, many small tests were performed along the way to ensure that each individual unit of the design was working. For example tests were set up to test the ADC as well as the UART capability. While these individual unit test were very important in making sure that everything worked as intended they will not be discussed in detail in this report. Instead comprehensive tests, which test the integrity of the whole system, including communication between different units, will be used to show proper functionality.

One very important unit test was testing the analog parts of the circuit. For the voltage regulation circuit, not much testing was needed because the circuit was designed using the schematic provided in the data sheet for the buck converter, which was previously depicted in the schematic for the circuit. On the other hand, the amplification circuit was tested and tweaked to meet our needs. Figure 18, below, shows the results of the amplification circuit. We can clearly see it is bringing the small voltage of about 100mV up to about 4.3V at the output. While this is only a gain of 43 rather than the theoretical 67.7, we did notice other weird behavior with the circuit, which could most likely be attributed to some slightly faulty soldering as well as the use of a breakout board mounted to a breadboard. Another factor, and the most likely one was that there was a large capacitive effect somewhere in the circuit which was causing the voltage of the output to stay at a rail for long periods of time. The combination of these factors resulted in the circuit sometimes sitting at either rail with no input or not amplifying properly. Nonetheless, when the circuit was not exhibiting this strange behavior it accomplished the goal of amplifying the voltage from the vibration sensor.

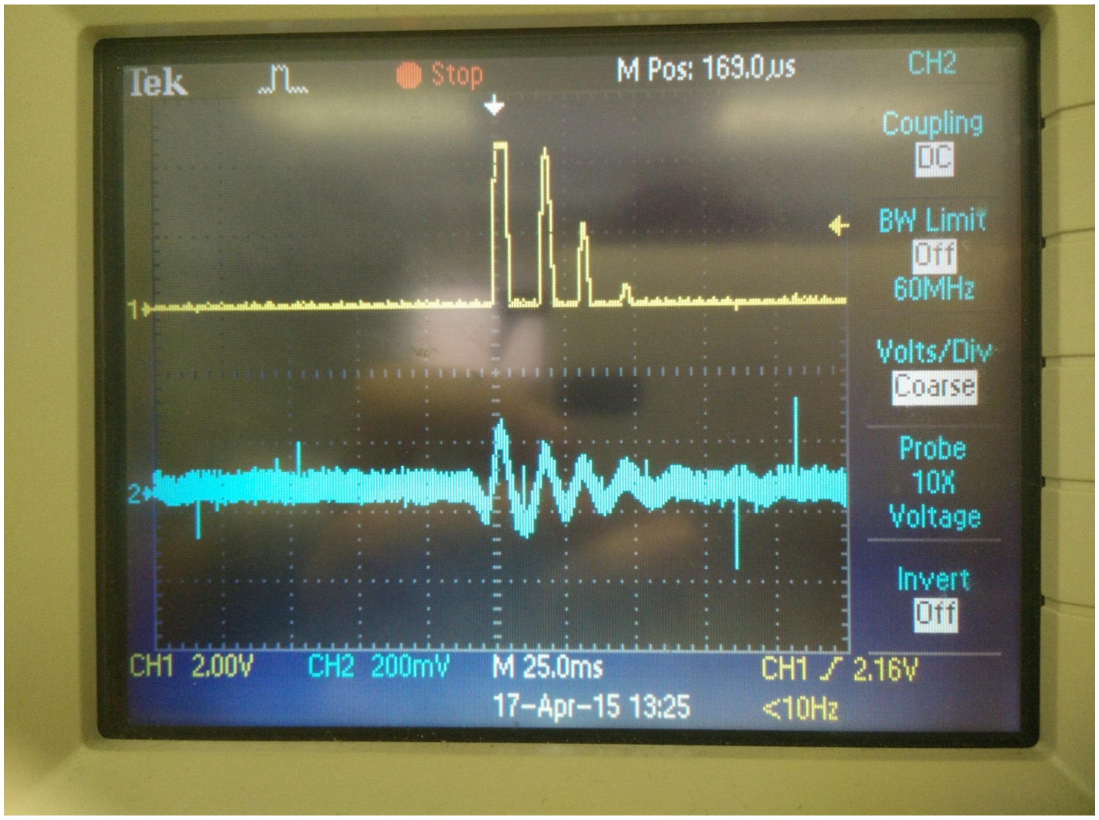


Figure #18: Test on Oscilloscope.

The next test we ran was testing that all the components of the MSP430 software were working properly. We began by connecting a function generator directly to the ADC of the MSP430 and connected to the UART ports of the MSP430 using PuTTY, a serial terminal interface. In this test we would turn the function generator on or off at random frequencies and amplitudes to try and get an understanding of what what range of frequencies and voltages the sensor would detect. If the sensor detected a signal change from on to off or off to on, it would send the most up to date state through the serial interface to the terminal. We found that anything above 20Hz was detected and that the minimum voltage necessary to consistently detect the presence of the signal was 1.5V, which was our original goal in terms of voltage. In terms of the minimum frequency, while we do not know exactly what frequency a furnace will generate, we can safely assume there is a significant portion above 20Hz since the hum is audible and the lowest frequencies humans can hear is 20Hz.

The second portion of this test was to make sure that the sensor could keep track of the time in 30-second segments, and write to and read from the flash memory. This was done by sending a certain character to the MSP430 through UART, which would prompt the MSP430 to dump the contents of its flash to the serial terminal. This test worked as intended and confirmed that the data we wanted was being accurately saved and read from the flash memory.

The final test was to make sure that the application worked properly. This was accomplished by making sure that the application could find the device, connect, and send and receive data, as well as process and graph this data. As previously stated, the team took an existing open source application and modifying it to our needs for the development of the application. As such, testing was done in two parts. The first part was testing the original Bluetooth terminal application and that data could be sent and received. This was fairly simple as we just replicated our above test, but replaced PuTTY with this Bluetooth terminal application. This worked as expected

The second half of the test was to make sure that the application could take data from the MSP430 and process and form a graph from it. This was done by generating random data in the same format, on the phone and feeding it into the graphing application. This test was successful as the graph was generated properly.

The final test of the systems before being placed on the PCB was a comprehensive test, and unfortunately it failed. The main problem was in the integration between the sensor and the application. For some reason most likely relating to Bluetooth LE, random characters were being added to the data being sent from the sensor to the application. While the sensor could accurately record and send the data, and the application could accurately process and graph the data, somewhere between the sensor and the app the data was being corrupted. We are currently looking into this, and feel that a solution should be fairly simple, but at this time no solution has been found.

2. PCB

The next set of tests involved getting the circuit working on the PCB. The manner in which we went about these tests was fairly similar to how we did the previous tests, but unfortunately we did not have time to go as in depth with the tests on the PCB.

The first test that we did was to make sure that the voltage regulator was delivering the proper voltage to all of the components on the board. This test unfortunately failed and halted many of the other tests which we would have liked to complete. The main problem with the power delivery on our circuit board was noise on the 3.3V output from the buck converter. While we are not exactly sure where this noise came from, we presume it may have come from the fact that our PCB was fabricated without using separate layers for the ground and 3.3V planes. This most likely would have reduced noise considerably, because when using the breadboard, no noise was exhibited with the same components. We now know for future reference to isolate the power and ground planes from the rest of the circuit when making a PCB in order to minimize the noise on both the power and ground traces.

While we weren’t able to revise the PCB and get a new revision delivered, we were able to filter the noise on the power plane and test other parts of the system on the breadboard, while being powered by the PCB. This was done by using a 100uF capacitor across the ground and power planes to eliminate any noise on the breadboard. Once this was done we were able to confirm some results that had been previously established by the earlier tests. The first is that the Bluetooth module and the MSP430 were communicating properly and could send and receive data reliably over UART. The second is that the once again, the data was getting corrupted when being processed by the application. This meant that we still could not get data to be sent from the MSP430 to our application to be graphed. While we know the application could graph the data reliably, we were not able to get the proper data to the application.

Conclusion:

Although the team ended up having a few unexpected problems that ended up prolonging the design process, the project as a whole turned out to be pretty successful. When it came to debugging the code it ended up being more difficult than we had foreseen. We ended up trying to find examples and help on the internet, but with Android Studio being in its infancy, this ended up being relatively difficult. As a project that we took from the initial concept stage to an almost fully realized product, we are happy with what we accomplished.

We successfully designed a sensor which could track vibrations and keep a log of the times when these changes in vibrations occurred. We were also able to create an application to connect to the sensor over Bluetooth and send and receive data from the sensor and graph it. There were two areas where we ended up falling short of ours goals. The first was getting the hardware to work properly on the PCB. This did not happen because of inexperience with PCB layouts amongst other things. The second area where we failed to complete our goal was in getting data to reliably be transferred between the application and the sensor. While this whole project proved to be more difficult than we had initially anticipated, and we were not able to fully achieve our goals, we are nonetheless happy with the outcome and how much we accomplished. Figure 18 below shows what ended up being the final deliverable for the sensor and application for this project.

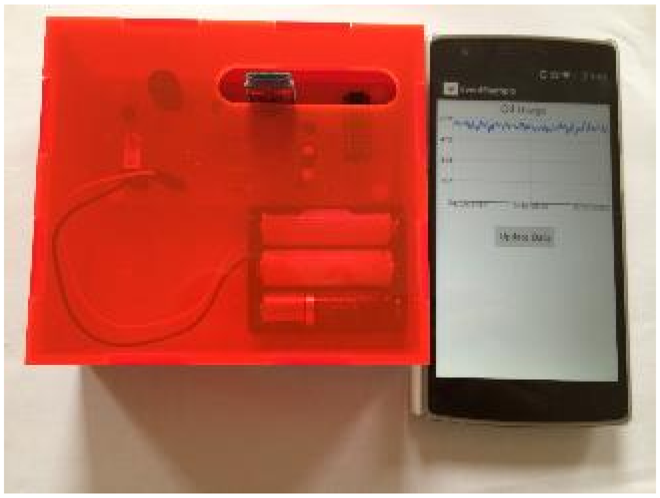


Figure #19: Phone Application Screen Shot and Sensor

Future Addition for another MQP Team:

After presenting the project idea and working on it for a few months the team was able to come up with a couple ideas have come up for future additions. With these ideas open many new possibilities for growth to take this product to the next level. These ideas are not all from the team involved with the project some are from the students and professors who attended the Energy IQP and MQP presentation throughout the year.

One idea that came up during one of the presentation days was that the sensor send out an email to the homeowner’s local oil provider to notify them that the tank was running low. This would be a super helpful and useful addition to add to the product. It also has the benefit of adding another buying point to the product. The only problem with the idea is it would have to transfer from Bluetooth to a Wi-Fi form of communication because of the need of the Internet to send the email.

One idea that we had originally proposed was the addition to be able to change the time scale of the graph so that the user had the ability to decide a week, month or years’ worth of data to view. This is still on the horizon for the extra time during D term but is lower on the priority list. To add this is just more coding within the application and maybe a MQP team with more coding experience would find this to be a very doable addition to add to the product.

For a final future suggestion would be to minimize the size of the final product while adding more powerful computing chip with more memory available. As with many products on the market today, as the new model is released it always seems that they are getting smaller and more powerful. This is something that could be accomplished very easily. During the building of the device the team came to notice about half way through B term that a processor with more flash would have been more beneficial, but the chip chosen still would work fine. Also the layout of the PCB is not condensed as much as is could be.

With these suggestion the team believes that there is definite room for improvement. There are many more ways to better and make this product more appealing to a consumer. With these ideas there is at least a good starting platform for another MQP team to possibly take over.

References:

CNG Units Explained. Retrieved April 28, 2015, from http://www.nat-g.com/why-cng/cng-units-explained/

Household Heating Costs. Retrieved April 28, 2015, from http://www.mass.gov/eea/energy-utilities-clean-tech/misc/household-heating-costs.html

Inside & out. Retrieved November 12, 2014, from https://nest.com/thermostat/inside-and-out/#explore-your-nest

Introduction to How to Repair Oil Furnaces - HowStuffWorks. Retrieved September 17, 2014, from http://home.howstuffworks.com/home-improvement/heating-and-cooling/how-to-repair-oil-furnaces.htm

Levi, M. (2010, June 11). Reducing U.S. Oil Consumption. Retrieved September 22, 2014, from http://www.cfr.org/oil/reducing-us-oil-consumption/p22413

Northeast braces for home heating oil increases. Retrieved September 20, 2014, from http://abcnews.go.com/Business/story?id=5270588

Ruede, D. Temperature@lert blog. Retrieved October 7, 2014, from http://www.temperaturealert.com/blog.aspx?CntTagID=f0e3f5dc-cabe-434a-8134-a13aee22e872

Seeed Growing the Difference. Retrieved September 20, 2014, from http://www.seeedstudio.com/depot/Piezo-Sensor-MiniSense-100-p-426.html

The Brooklyn Cooperator. (2010, October 19). Retrieved October 7, 2014, from http://bkcoop.blogspot.com/2010/10/natural-gas-vs-heating-oil.html

Uses of Natural Gas. Retrieved September 17, 2014, from http://geology.com/articles/natural-gas-uses/

Appendix A:

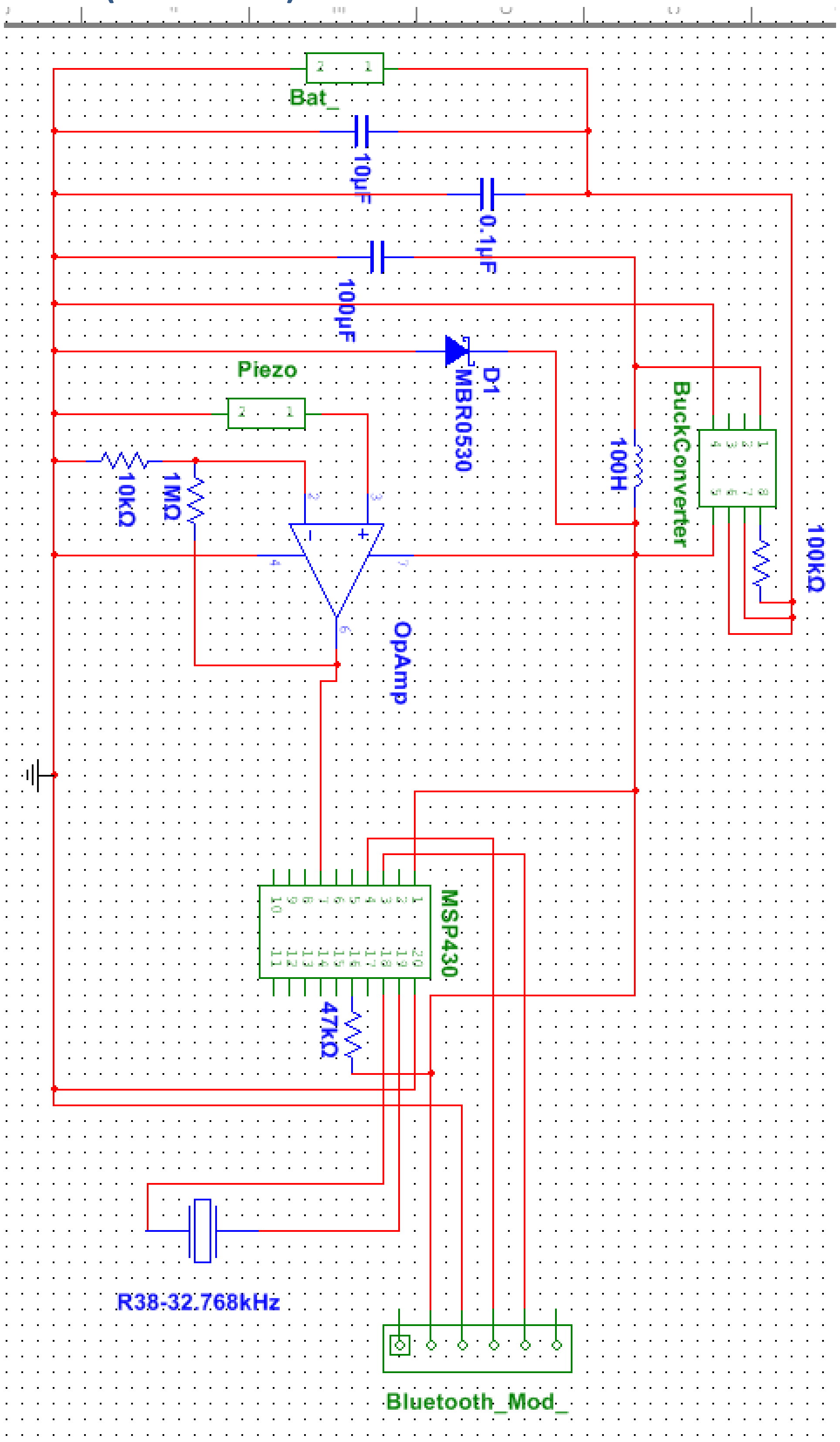
Hardware Code:

```
/* Example code demonstrating the use of the hardware UART on the MSP430G2553
to receive
 * and transmit data back to a host computer over the USB connection on the
MSP430
 * launchpad.
 * Note: After programming it is necessary to stop debugging and reset the uC
before
 * connecting the terminal program to transmit and receive characters.
 * This demo will turn on the Red LED if an R is sent and turn it off if a r
is sent.
 * Similarly G and g will turn on and off the green LED
 * It also transmits the received character back to the terminal.
 */

#include "msp430g2553.h"
#define LED0 BIT0
#define LED1 BIT6
#define ON 1
#define OFF -1

void UARTSendArray(unsigned char *TxArray, unsigned char ArrayLength);
void flashSave(signed long int elapsed);
void readFlash();
void clearFlash();
void ConfigureAdc(void);
char* itoa(long value, char* result, int base);
void write_SegC (long fValue);
void copy_C2D (void);
void erase(void);

long fValue;
char *Flash_ptrC = (char *) 0x1040;
char *Flash_ptr = (char *) 0x1040;
static unsigned char data;
static unsigned int resolution = 30;
static long int ticks=0;
static long int flashTest = 0;
static unsigned     int tempSecs = 0;
static char aTime[10];
int setTime=0;
int timeCount=0;
unsigned int value=0;
static signed char state;
static signed char tempState;

void main(void)

{
       WDTCTL = WDTPW + WDTHOLD; // Stop WDT
	P1DIR |= LED0 + LED1;
	P1OUT &= ~(LED0 + LED1);



	BCSCTL1 = CALBC1_1MHZ; // Set DCO to 1MHz
	DCOCTL = CALDCO_1MHZ; // Set DCO to 1MHz

	/* Configure hardware UART */
	P1SEL = BIT1 + BIT2 ; // P1.1 = RXD, P1.2=TXDs
	P1SEL2 = BIT1 + BIT2 ; // P1.1 = RXD, P1.2=TXD
	P1SEL |= BIT5;
	UCA0CTL1 |= UCSSEL_2; // Use SMCLK
	UCA0BR0 = 104; // Set baud rate to 9600 with 1MHz clock (Data Sheet 
15.3.13)
	UCA0BR1 = 0; // Set baud rate to 9600 with 1MHz clock
	UCA0MCTL = UCBRS0; // Modulation UCBRSx = 1
	UCA0CTL1 &= ~UCSWRST; // Initialize USCI state machine
	IE2 |= UCA0RXIE; // Enable USCI_A0 RX interrupt


	ConfigureAdc();
	TACCTL0 = CCIE;                          // TACCR0 interrupt enabled
	TACCR0 =8191;
	TACTL = TASSEL_1 + MC_1;                 // ACLK, continuous mode

	FCTL2 = FWKEY + FSSEL0 + FN1;            // MCLK/3 for Flash Timing 
Generator
	fValue = 0x506F7065;

	erase();
	//clearFlash();

	__bis_SR_register(LPM0_bits + GIE); // Enter LPM0, interrupts enabled
}

#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A0_ISR(void)
{
	if (tempSecs==(40-1)){
		int tempValue;
		tempSecs = 0;
		ticks ++;
		ADC10CTL0 |= ENC + ADC10SC;
		__bis_SR_register(CPUOFF + GIE);
//		for(i=1;i<1000;i++){
//			tempValue = ADC10MEM;
//			if(tempValue>value)
//				value = tempValue;
//		}
		tempValue = ADC10MEM;
		if(tempValue>value)
			value = tempValue;
		if (value>300)
		{
				P1OUT &= ~(LED0 + LED1);
				P1OUT ^= LED0;
				tempState = ON;
				//UARTSendArray("H", 2);
			}
			else
			{
				P1OUT &= ~(LED0 + LED1);
				P1OUT ^= LED0;
				tempState = OFF;
				//UARTSendArray("L", 2);
			}
			value = 0;
			if (tempState != state)
			{
//				switch (tempState){
//				case(ON):{
//					UARTSendArray("On", 2);
//				}
//					break;
//				case(OFF):{
//					UARTSendArray("Off", 3);
//				}
//					break;
//				default:
//					break;
//				}


				state = tempState;
				write_SegC((long)state*ticks*1000);
			}
		}
		else if  (tempSecs==(40-2)){
			int tempValue;
			ADC10CTL0 |= ENC + ADC10SC;
			__bis_SR_register(CPUOFF + GIE);
			tempSecs++;
			tempValue = ADC10MEM;
			if(tempValue>value)
				value = tempValue;
		}
		else if  (tempSecs==(40-3)){
			int tempValue;
			ADC10CTL0 |= ENC + ADC10SC;
			__bis_SR_register(CPUOFF + GIE);
			tempSecs++;
			tempValue = ADC10MEM;
			if(tempValue>value)
				value = tempValue;
		}
		else if  (tempSecs==(40-4)){
				int tempValue;
				ADC10CTL0 |= ENC + ADC10SC;
				__bis_SR_register(CPUOFF + GIE);
				tempSecs++;
				tempValue = ADC10MEM;
				if(tempValue>value)
					value = tempValue;
			}
		else{
			tempSecs++;
		}



}


// Echo back RXed character, confirm TX buffer is ready first
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
	data = UCA0RXBUF;

	//UARTSendArray("Received command: ", 18);
	//UARTSendArray(&data, 1);
	//UARTSendArray("\n\r", 2);

	switch(data){
	case 'R':
	{

		//char* itoaResult = (char*)malloc(15);
		P1OUT ^= LED0;
//		write_SegC((long)fValue);			// Write segment C,
increment value
		   //copy_C2D();
		//itoaResult = itoa(0x506F7065, itoaResult, 10);
		//UARTSendArray(itoaResult, 15);
		UARTSendArray("1000\n", 5);
		UARTSendArray("-2000\n", 6);
		UARTSendArray("3000\n", 5);
		UARTSendArray("-4000\n", 6);
		UARTSendArray("5000\n", 5);



	}
	break;
	case 'r':
	{
		P1OUT &= ~BIT0;
	}
	break;
	case 'f':
		{
			//readFlash();
		}
```

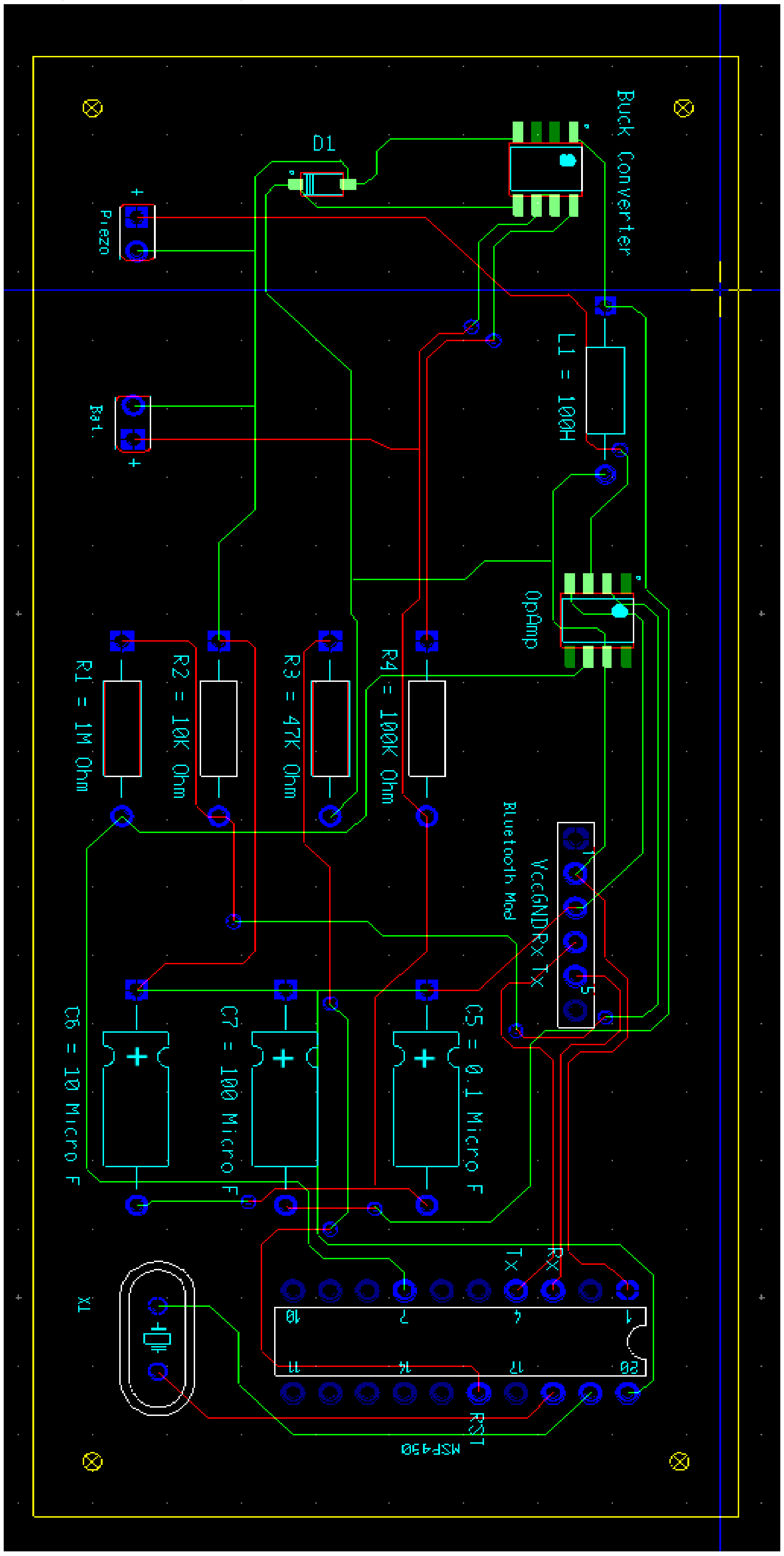
```
			break;
		default:
		{
			//UARTSendArray("Unknown Command: ", 17);
			//UARTSendArray(&data, 1);
			//UARTSendArray("\n\r", 2);
		}
		break;
		}

}

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR (void)
{
	__bic_SR_register_on_exit(CPUOFF);
}


void UARTSendArray(unsigned char *TxArray, unsigned char ArrayLength){
	// Send number of bytes Specified in ArrayLength in the array at using
the hardware UART 0
	// Example usage: UARTSendArray("Hello", 5);
	// int data[2]={1023, 235};
	// UARTSendArray(data, 4); // Note because the UART transmits bytes it
is necessary to send two bytes for each integer hence the data length is twice
the array length

	while(ArrayLength--){ // Loop until StringLength == 0 and post decrement
		while(!(IFG2 & UCA0TXIFG)); // Wait for TX buffer to be ready for
new data
		UCA0TXBUF = *TxArray; //Write the character at the location
specified py the pointer
		TxArray++; //Increment the TxString pointer to point to the next
character
	}
}



// Function containing ADC set-up
void ConfigureAdc(void)
{
	ADC10CTL1 = INCH_5 + ADC10DIV_3 ;
	ADC10CTL0 = SREF_0 + ADC10SHT_3 + ADC10ON + ADC10IE;
	ADC10AE0 |= BIT5;
}


//void clearFlash()
//
//{
//
//	_DINT();
```

```
//	while(BUSY & FCTL3);
//	FCTL2 = FWKEY + FSSEL_1 + FN3;             // MCLK/3 for Flash Timing
Generator
//	FCTL1 = FWKEY + ERASE;                               // Clear LOCK bits
//	FCTL3 = FWKEY;                 // set erase bit;
//	char *addrErase = (char*) 0x0E000;
//	*addrErase = 0;
//	addrErase = (char*) 0x0E020;
//	*addrErase = 0;
//	while(BUSY & FCTL3);
//	FCTL1 = FWKEY;                               // Clear WRT bit
//	FCTL3 = FWKEY + LOCK;              // Set LOCK & LOCKA bit
//	_EINT();
//
//}
//
//void readFlash(){
//	unsigned int i;
//	signed long int output;
//	unsigned char toggle = 0;
//	int upper;
//	int lower;
//	int *Read_ptr = (int*)0x0E000;
//	/*for(i=0; i<128; i++){
//		if (toggle==0){
//			upper = *Read_ptr;
//			toggle++;
//			Read_ptr++;
//		}
//		else if (toggle ==1){
//			lower = *Read_ptr;
//			toggle--;
//			Read_ptr++;
//			output = lower + ((long int)upper<<16);
//			char str[10];
//			snprintf(str, 10, "%d", output);
//			UARTSendArray(str, 11);
//			UARTSendArray("\n\r", 2);
//		}
//	}*/
//	//upper = *Read_ptr;
//	//Read_ptr++;
//	//lower = *Read_ptr;
//	//char up[2];
//	//up = "1";
//	//char low[2];
//	//low = "1";
//	UARTSendArray("N", 1);
//	UARTSendArray("O", 1);
//	UARTSendArray("\n\r", 2);
//
//	//clearFlash();
//}

void erase(){
      Flash_ptr = (char *) 0x1040;              // Initialize Flash pointer
      FCTL1 = FWKEY + ERASE;                    // Set Erase bit
      FCTL3 = FWKEY;                            // Clear Lock bit
      *Flash_ptr = 0;
      FCTL1 = FWKEY;                            // Clear WRT bit
      FCTL3 = FWKEY + LOCK;
}


void write_SegC (long fValue)
{
  //char *Flash_ptr;                          // Flash pointer
  //unsigned int i;

 if(Flash_ptr==(char*)0x1080){
        erase();
 }
//  Flash_ptr = (char *) 0x1040;              // Initialize Flash pointer
//FCTL1 = FWKEY + ERASE;                    // Set Erase bit
FCTL3 = FWKEY;                            // Clear Lock bit
//  *Flash_ptr = 0;                           // Dummy write to erase Flash
segment

  FCTL1 = FWKEY + WRT;                      // Set WRT bit for write operation

  char first, second, third, fourth;
  first = (char) fValue;
  second = (char) (fValue>>8);
  third = (char) (fValue>>16);
  fourth = (char) (fValue>>24);
//     first = 'e';
//    second = 'p';
//    third = 'o';
//    fourth = 'P';



    *Flash_ptr++ = fourth;                    // Write value to flash
    *Flash_ptr++ = third;
    *Flash_ptr++ = second;
    *Flash_ptr++ = first;

  FCTL1 = FWKEY;                            // Clear WRT bit
  FCTL3 = FWKEY + LOCK;                     // Set LOCK bit
}

void copy_C2D (void)
{
                          // Segment C pointer
  //char *Flash_ptrD;                         // Segment D pointer
  unsigned int i;
  char first, second, third, fourth;
  long result;
  char* itoaResult = (char*)malloc(15);
  Flash_ptrC = (char *) 0x1040;

//  if(Flash_ptrC == (char*)0x1080){
//      Flash_ptrC = (char *) 0x1040;             // Initialize Flash segment
C pointer
//  }

  //Flash_ptrD = (char *) 0x1000;              // Initialize Flash segment D
pointer
  FCTL1 = FWKEY + ERASE;                     // Set Erase bit
  FCTL3 = FWKEY;                             // Clear Lock bit
  //*Flash_ptrD = 0;                           // Dummy write to erase Flash
segment D
  FCTL1 = FWKEY + WRT;                       // Set WRT bit for write operation

      while(Flash_ptrC < Flash_ptr){
             fourth = *Flash_ptrC++;           // copy value segment C to
segment D
             third = *Flash_ptrC++;
             second = *Flash_ptrC++;
             first = *Flash_ptrC++;
             result =
((long)fourth<<24)+((long)third<<16)+((long)second<<8)+((long)first);
             itoaResult = itoa(result, itoaResult, 10);
             UARTSendArray(itoaResult, strlen(itoaResult));
             UARTSendArray("\n", 1);
      }

      free(itoaResult);
      erase();
      ticks = 0;

  FCTL1 = FWKEY;                             // Clear WRT bit
  FCTL3 = FWKEY + LOCK;                      // Set LOCK bit
}



//void flashSave(signed long int elapsed)
//{    int lower = 0x00;
//     int upper = 0x00;
//     lower += (int) elapsed;
//     upper += (int)(elapsed>>16);
//     if(Flash_ptr < (int *)0xFFC0){
//     _DINT();                              // Disable interrupts(IAR
workbench).
//     FCTL2 = FWKEY + FSSEL_1 + FN2; //set clock
//     FCTL3 = FWKEY; //unlock
//     FCTL1 = FWKEY + WRT; //enable writing a single word
//     *Flash_ptr++=upper;
//     *Flash_ptr++=lower;
//     while(BUSY & FCTL3);
//     //clear wrt bit
//     FCTL1 = FWKEY;
//     FCTL3 = FWKEY + LOCK; //lock  the memory.
//      _EINT();
//      }
//}




char* itoa(long value, char* result, int base) {
             // check that the base if valid
             if (base < 2 || base > 36) { *result = '\0'; return result; }

             char* ptr = result, *ptr1 = result, tmp_char;
             int tmp_value;

             do {
                    tmp_value = value;
                    value /= base;
                    *ptr++ =
"zyxwvutsrqponmlkjihgfedcba9876543210123456789abcdefghijklmnopqrstuvwxyz" [35
+ (tmp_value - value * base)];
             } while ( value );

             // Apply negative sign
             if (tmp_value < 0) *ptr++ = '-';
             *ptr-- = '\0';
             while(ptr1 < ptr) {
                    tmp_char = *ptr;
                    *ptr--= *ptr1;
                    *ptr1++ = tmp_char;
             }
             return result;
      }
```

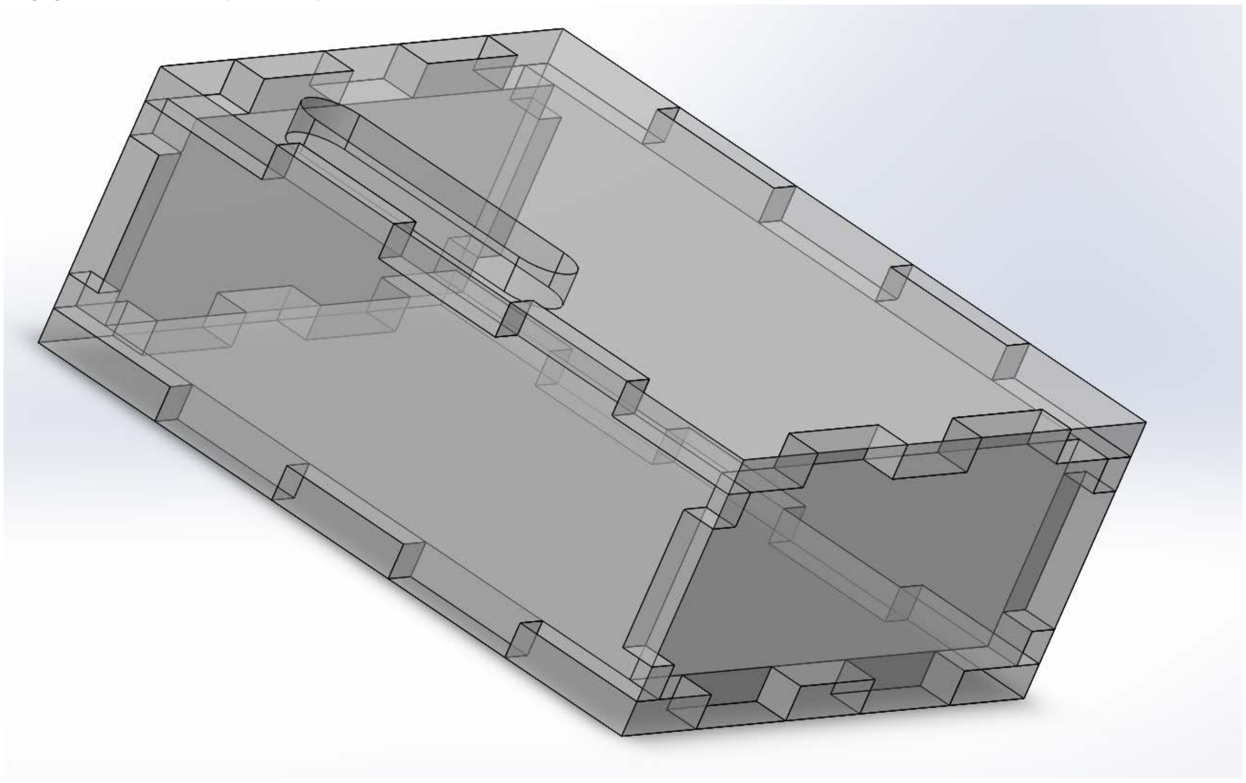
Appendix B (schematics):



Appendix C (PCB Board):



Appendix D (Case):



Appendix E (User Manual):

Oil Detection Sensor:

Thank you! You have purchased the wireless Bluetooth connective Oil Detection Sensor to help your family track and discover what your house holds patterns are in their home heating habits. This device has an application for your android smart phone that is connected to the sensor that will be installed on your furnace through Bluetooth 4.0 BLE Connectivity.

Customer Requirements:

- Android or Apple smart phone
- 3 AA batteries
- Oil Furnace

Installation Instructions:

This device has a very simple installation plan for the everyday do-it-yourselfer. Before attaching the device to the furnace it is required that 3 AA batteries are installed. Please remove the cover of the sensor to expose where the batteries must be installed. ***Note:***

make sure that the batteries are installed in the proper position before re-installing the cover of the sensor.

Next you should identify a free space on the outer metal protective cover for the furnace. Make sure that there are no obstructions or obstacles roughly 2 inches around the sensor. The sensors shell has four magnets attached to it and only requires the user to place on the metal housing of the furnace. ***Note****: for the best results place the sensor as close to the top of one of the four side of the furnace as possible.*

Sensor installation is now complete.

Downloading the Application:

The first step is to search for the application in the mobile application store. Then Click download. Open the application and begin the process of connecting you phone to the sensor device.

Trouble Shooting:

Problem:	Possible Solution:
Device not Turning on.	Dead Batteries, Battery pack may not connected.
Phone not retrieving data.	Bluetooth is not ON on phone, Phone is out of range, device is not ON,
Missing Data transfer.	Bluetooth transmission was cut during sync time, memory was filled from the last sync so there may be a date gap in the data that was received.