BOEING
NFC PART VERIFICATION SYSTEM

ECE 445

Jinjoo Nam - Alper Olcay - Vigneshwar Karthikeyan

TA: Kevin Bassett
May 1, 2013

Abstract

The NFC Part Verification system is designed to help Boeing digitalize paper trails and eliminate the possible usage of the parts that have been damaged during transportation or underwent faulty industrial processes. With NFC, a more secure system can be designed since it requires the user to be in close proximity of a NFC enabled tag for a successful read. The system demonstrates the usefulness of NFC technology in the case studies, namely the part tracking case study and the heat treatment case study. Both case studies make use of Arduino microcontrollers as well as different kinds of sensors to determine treatment and transportation conditions.

Table of Contents

1. Introduction

1.1 Project Overview

Our main goal was to develop a viable alternative for paper trails in part tracking and verification, for our client Boeing. In order to accomplish our goal, we integrated NFC technologies into two case studies, namely the part tracking case study and the heat treatment case study. For our part tracking case study, we have created a sensor bundle which would go on each and every part, as well as a NFC control station. For our heat treatment case study, we have developed a "smart" furnace that takes the employee through a heat treatment process step by step with the help of NFC and a LCD. Listed below are the main benefits and features of our project.

Benefits:

- Cheaper than existing RFID technologies.
- More secure due to shorter range compared to other wireless technologies.
- Complete digitalization of paper trails.
- Ease of part and process tracking.

Features:

- 13.56 MHz is the operating frequency of the system.
- Sensor bundle consisting of a digital temperature sensor and a 2 axis accelerometer for the part tracking case study.
- Expected tag operating range of 2-6cm's.
- Fully functional visual aid system with LCD.

1.2 High-Level Design

The first step in an aircraft part's lifetime is logistics and arrival into the Boeing facilities. In order to track the transportation and storage conditions of the part, we have developed a sensor bundle which interacts with an Arduino board. The sensor bundle constantly measures temperature and acceleration values and stores them into a SD card that is placed in the Arduino SD Card Shield. In order to avoid cluttering of data, Arduino only logs the data onto the SD card when the actual measurement is not in the tolerated range set up by Boeing. When a particular part arrives into a Boeing processing plant, an employee goes through the data on the SD card and uses the NFC station to write a quality assurance check mark onto a NFC enabled tag, and places that tag onto the part. With that, the part now has identification, and can move downstream to other processes such as the heat treatment facility. Our complete block diagram for this case study is shown in Figure 1.

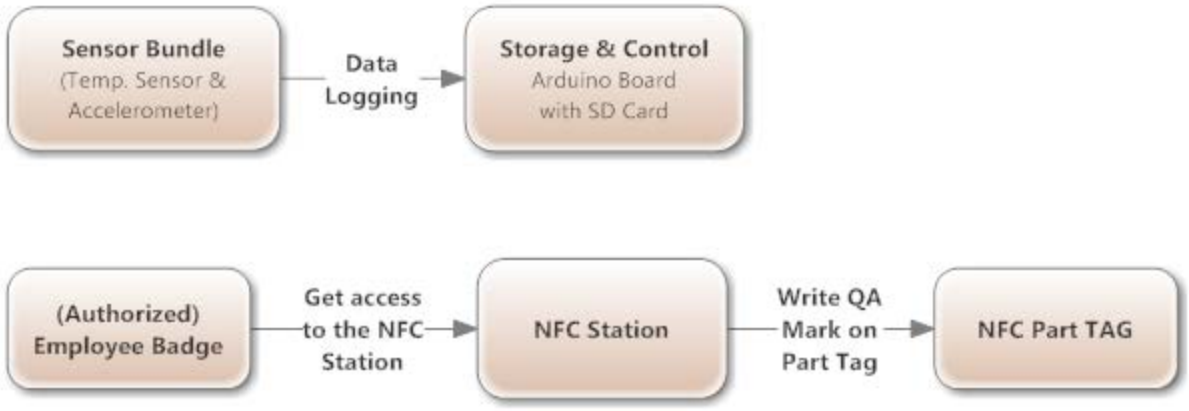

Figure 1: High level block diagram for Part Tracking Case Study.

After going through the first quality assurance check, the part moves onto a treatment facility. As a result of extensive discussions, we have decided that simulating a heat treatment facility would be the most logical choice. In this case, the Arduino is used as a control block, as well as a data transmitter. A thermistor circuit measures temperature constantly and this information is transferred to an online cloud storage system. The role of NFC is to provide security by prompting the user to scan their ID badge, as well as placing another quality assurance checkmark onto the part tag when the process is completed. Figure 2 shows a complete block diagram of this case study.

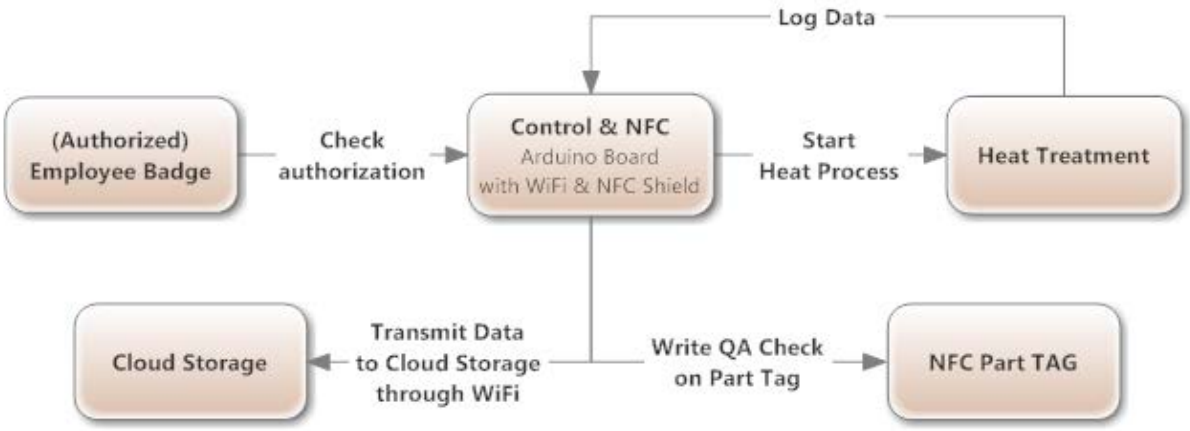

Figure 2: High level block diagram for Heat Treatment Case Study.

1.3 Design Revisions

Since our project was designed for a client, it underwent multiple changes and revisions during its development process. Most of the revisions that we decided to make were minor, such as switching to 9V batteries. However, there were major changes as well. Elaboration of the revisions is made in the following two sections.

1.3.1 Part Tracking

The part tracking case study underwent major revisions in the last four months. First and foremost, the case study contained a smartphone as a dedicated NFC reader. However, due to Boeing's policy about smartphones at the workplace and other difficulties, the idea of having a smartphone as a reader and designing an Android app was abandoned.

On top of that, due to storage issues, a SD Card shield for Arduino had to be used. This change led to additional coding as well as a clashing of pins with the WIFI shield for Arduino, which disabled our ability to send data wirelessly.

A number of minor changes were also made including but not limited to increasing battery voltage, changing the type of temperature sensor and accelerometer that was used, and separating the NFC unit from the sensor bundle.

1.3.2 Heat Treatment

Similar to the part tracking case study, the heat treatment case study originally contained a smartphone aspect which was then scrapped due to reasons stated in the section above.

Another major change in this case study was the fixing of the PCB. Our original PCB design for this case study involved an incorrect input voltage, which led to non-functionality. The design was then revised to produce a working PCB.

The heat treatment case study was originally designed to contain a control circuit which would be able to cut off current to the heater, which would effectively end the heating process. However, due to difficulties in getting a magnetic relay working, and the needs of our client, a lighter control method was set up, which involves using a LCD to visually aid employees throughout the process.

Some of the minor changes that were done in this case study are increasing supply voltage, delocalization of thermistors to get a more accurate temperature reading, and the block orientation of NFC badges and tags.

2. Design

2.1 Part Tracking

2.1.1 Sensor Bundle Unit

The temperature sensor is capable of measuring temperatures in the range of -40 to 120 ℃ with an accuracy of $\pm$ 1 ℃. The accelerometer is capable of measuring up to $\pm 3G$. The sensor bundle is powered up from the 5V pin on the Arduino and is tied to the common ground with it.

2.1.2 Storage Unit

SD card shield for the Arduino Uno was used for storage capabilities. During the process of transportation, the SD card is written to when the control unit has determined that conditions for the part are no longer ideal. Therefore, if the part is not damaged during any part of transportation, the SD card will arrive at the end of the process with no data having been stored.

2.1.3 Control Unit

The Arduino Uno was used as the control unit. It is powered up with a 9V Energizer battery. The microcontroller is programmed to be able to read the sensor inputs from the Sensor Bundle unit. The data from both the Temperature Sensor and Accelerometer is read and depending on the range, the data is written to an SD card. For temperature values, the microcontroller is programmed to check if the temperatures fall in the range of 20 to 26 ℃ and if they fall outside that range, the data is written to the SD card. For values that fall into that ideal range, the data is discarded. For acceleration values, the sensor input is checked for the range of $\pm$ $0.5G$. Sensor inputs within that range are discarded but values outside that range will be logged to the SD card. The code written for the Arduino Uno to perform this control is provided in the Appendix.

2.1.4 NFC Station Unit

The NFC Station consists of an Arduino Uno along with an Adafruit NFC Shield attached. The unit also has an LCD which functions to show the user instructions for completing the process. The NFC Shield stacked on the Arduino allows for the reading and writing capabilities to NFC tags, which will either be Employee Badges or Part Tags.

The NFC Badge is a Mifare Classic NFC card. The badge will contain the relevant information about the employee including their employee ID and access privileges. The NFC card has multiple data blocks for storage with each data block consisting of 2 bytes. The NFC card will hold the Employee ID in data block 6 while data block 5 will hold their access privileges. Before accessing the NFC station, the employee would have to scan their badge on the NFC shield and if they are deemed to have access, would be able to proceed to the next stage in the quality assurance process.

The NFC part tag is a Mifare Classic 1k NFC tag. The NFC Tag has multiple data blocks for storage of 2 byte words. For both case studies, the Part tags use data block 5 in order to store the part's serial

number and the data block 6 to store whether or not a quality check for a particular industrial process has been completed.

The station functions to first prompt the user to scan their employee badge and the instruction “Scan Badge” is displayed on the LCD. Once the user scans their badge, based on the access privileges on the badge, the user will either be prompted to scan a different badge if credentials were not validated or the user will be allowed to proceed to the next step if credentials are verified. Upon verification, the user will be able to remove the SD card to plug into a PC and read the data on the card. The NFC station will then ask the user to scan the Part Tag of the part. Once the user has completed the quality assurance step, they would scan the Part Tag on the NFC shield which would write a quality assurance mark onto the Part Tag. The NFC station would then be reset for the next Part that needs to be checked.

2.2 Heat Treatment

2.2.1 Heat Treatment Unit

The Heat Treatment Unit consists of the PCB design as shown in Figure 7. The Schematic is shown in Figure 6 As seen from the schematic; the circuit consists of two negative feedback operational amplifier circuits that contain the thermistor.

$$R_T = R_o e^{\beta\left({}^1/_T - {}^1/_{T_o}\right)}$$

Equation 1: The thermistor equation

The heat treatment unit also consists of a wooden box of dimensions 10 by 10 by 10 inches to simulate an industrial furnace. A hot plate which is powered by an outlet, is placed inside the box to allow for heating of the part. The unit is powered up by 12V from a bench outlet.

The thermistors are delocalized and placed inside our heat treatment box so that temperatures changes that the part undergoes within the furnace can be monitored.

2.2.2 NFC & Control Unit

The control unit for this case study is composed of the Arduino Uno along with an Adafruit NFC Shield, Adafruit Wifi Shield and an LCD. The Arduino Uno is able to take sensor inputs from the PCB and determine what conditions are met during the heat treatment process. It is powered from a bench supply.

The Arduino will first use the Wifi Shield to connect to a Wifi Hotspot created by a Samsung Galaxy SII with a password enabled connection. Once connected, the Arduino will then prompt the first step the employee has to take to start the process, which is scanning the NFC Badge.

The Employee Badge for this case study is a Mifare Classic NFC card. The badge will contain the relevant information about the employee including their employee ID and also whether or not they have access to particular processes in the manufacturing plant. The NFC card has multiple data blocks for storage with each data block consisting of 2 bytes. The NFC card will hold the Employee ID in data block 6 while data block 5 will hold their access privileges. Before accessing the treatment circuit, the employee would have to scan their badge on the NFC shield in if they are deemed to have access, would be able to proceed to the next stage in the heat treatment process.

The other NFC component is a part tag which is a Mifare Classic 1k NFC tag. The NFC Tag has multiple data blocks for storage of 2 byte words. For both case studies, the Part tags use data block 5 in order to store the part's serial number and the data block 6 to store whether or not a quality check for a particular industrial process has been completed.

In order to start the process, an employee would be prompted by the LCD to "Scan Badge" and the employee would have to scan their employee badge. If their credentials are verified, the next message the user is prompted to scan the part tag associated with the part that will then be put into the furnace for the heat treatment process. Once the part tag has been scanned the user will be able to put the part into the heat treatment box and turn on the hot plate. With the serial number acquired from the Part Tag, the Arduino will then start to collect the sensor input from the PCB. The Arduino then makes a connection using the Wifi Connection the our Cloud Storage Unit, COSM where a put request will add the temperature values being acquired to a data stream that is named in accordance with the part serial number. In this way, an employee can check the temperature profile a part has been put through based on its part serial number.

During the data logging process, there is also control implemented to check the range of temperatures. When the temperature reaches 50 ℃ the user will be prompted by the LCD to turn off the hot plate and remove the part because the part has undergone the necessary heat treatment profile. Until this check is achieved, the control will continue to log data to COSM. After the part has been removed, the user will be prompted to again scan the employee badge in order to place a successful quality assurance mark on the Part Tag. If however, by some error, the user does not remove the part at the correct time, and the temperature the part reaches exceeds 80 ℃, the LCD will indicate a failure of the process and when the Part Tag is scanned on the NFC shield, a quality mark will be placed on the tag indicated a failed process.

Once the steps have been completed, the Arduino is then reset for the next part that is to be put through the heat treatment process.

2.2.3 Cloud Storage Unit

For streaming data wirelessly to a storage system, COSM was used. COSM is an online database service that uses an Application Programming Interface (API) to provide most of its functionality for acquiring and managing streaming data. COSM was used to simulate the Backend Server capabilities for the project.

3. Requirements & Verifications

Our original requirements had to be modified in the light of the revisions we made in our projects. Our current requirements include an accuracy tolerance for our sensors as well as accessibility ranges for NFC badges and tags. A complete Requirements & Verifications table is included in Appendix A.

3.1 Testing

We have conducted standard testing procedures in order to determine basic requirements such as battery voltages, NFC tag access ranges. On top of that, every part that was soldered on either a PCD or an Arduino shield went through extensive continuity checks with a standard hand held multimeter in order to make sure that the connections are made right.

We have not had the opportunity to prototype our PCB's due to the nature of our parts, however, we were able to conduct a small scale thermistor circuit on a breadboard to verify that our thermistors were in working condition. The circuit is shown in Figure 3.

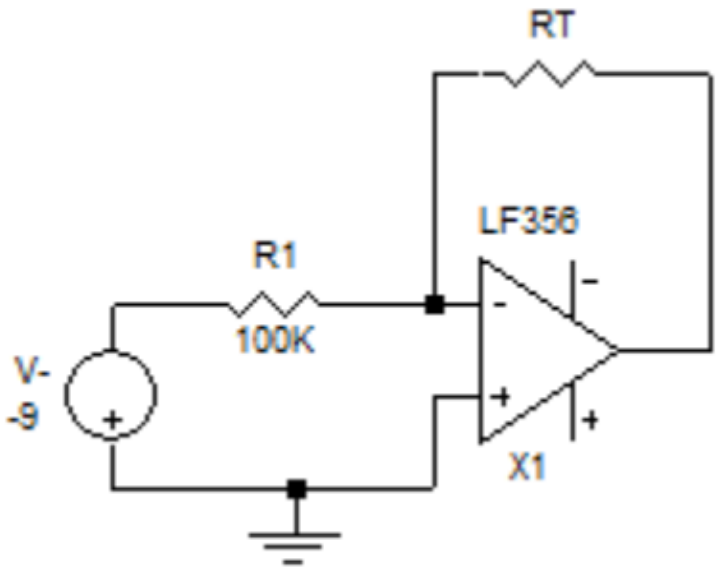

Figure 3 : Thermistor Test Circuit

With the help of this circuit, we were able to test and verify that our thermistors behaved as they were supposed to.

Due to the fact that our project was mainly software based, we have not had any specialized hardware testing methods besides the ones that are listed above.

3.2 Explanations of Failed Verifications

We were able to verify each of our requirements but one. The thermistor circuit which was mentioned in 3.1 was originally required to have a maximum variance of ±1 degrees Celcius. However, the thermistor output displayed errant behavior, and while sensing the changes in ambient temperature, its voltage output varied a lot (±2 degrees in room temperature). This is caused by the current state of the PCB that was used in the project, as the carving of some traces as well as adding solder junctions to solder jumper wires led to current leakages. On top of that, having bare metal parts in a testing environment filled with electronic devices possibly caused some 60Hz interference.

4. Cost

We were able to complete our project by using 10% of the funds Boeing has allocated for us. Even though we had to buy parts that we had not planned buying before for multiple contingency plans, the overall cost for Boeing still came out to be reasonable.

4.1 Parts

Even though we ended up not buying a smartphone for this project, the cost of parts ended up in the same range with the estimation in our design review. Buying multiple Arduino boards and their shields effectively increased the costs. A detailed table with the cost of each and every part that was used in the project is included in Appendix B.

4.2Labor

We originally estimated that each member of our group would put an estimated 150 hours of work into this project. However, our estimations in this regard came up short as we spent around 300 hours each, which effectively doubled the labor costs. A detailed table is provided in Appendix B.

5. Conclusion

After 4 months of research and design, we were able to come up with two case studies that satisfy both our class requirements and the demands of our client. For each of our case studies, we were able to demonstrate the possible uses or shortcomings of NFC implemented into the lifecycle of a part. The part tracking case study provided a way to track the logistics process of a part and verify that the part has not been tampered with while the heat treatment case study simulated the use of NFC technologies in a “smart” furnace.

5.1 Accomplishments & Challenges

In general, our project was very successful, as we were able to demonstrate overall functionality as well as room for further development. For our heat treatment case study, we were able to successfully design a PCB, and simulate a heat treatment facility, as well as demonstrating read and write capabilities with an Arduino UNO R3 board equipped with a WIFI and a NFC shield. We were also able to use Cosm as a storage system for our streaming data.

The part tracking case study was also a success, as we successfully logged temperature and acceleration readings that are out of the desired range, onto a SD card. Another accomplishment in regards to this case study was configuring a NFC unit with LCD to prompt the user to scan their employee badge and the part tag in order to place a quality assurance check mark on the tag.

The first and foremost challenge we have encountered was designing a control circuit which would automatically cut off power from the heaters for our heat treatment case study. After trying out different relays and circuit configurations, we noticed that the Arduino board cannot output enough current to successfully drive a relay.

Another major challenge was to get the SD card shield and the NFC shield to work together in our part tracking case study. Due to a clashing of pins, we were not able to establish WIFI connectivity to a hotspot while the SD card shield was present on the Arduino. We overcame that challenge by separating part tracking case study into two units, namely the sensor bundle unit and the NFC unit. This particular separation also proved to be valuable for Boeing, as including a NFC chip on each and every part would be too costly.

5.2 Future Work

Due to the nature of our project, there will always be room for development and new ideas. The first and foremost improvement that can be made right away is to replace the Arduino board with a commercially available microprocessor. While essentially performing at the same level, such a microprocessor would be much smaller in size. Also a dedicated memory can be added to the circuit instead of the SD card.

Another area suitable for development is saving more time for Boeing employees. The current part tracking system we implemented required the employee to manually go through the contents of the SD card and then determine the condition of the part, which could be very time consuming. To streamline this process, we will implement a system which will let the employee know whether the SD card has data logged or not beforehand, by providing visual aid in the form of LED's. Therefore the employee would only control the SD cards that have data logged onto them, saving massive amounts of time and effort.

Finally, to make both of our case studies into a NFC system that can be actually used in a logistics process and a manufacturing plant, we must add proper enclosement for our devices as well as PCB's with silkscreen and the ability to work on a smaller battery. While most of these are cosmetic changes, they drastically increase the chance our project actually being implemented by Boeing.

5.3Ethical Considerations

We agree and abide to the IEEE code of Ethics. We followed the rules listed that applied to our project.

1. To accept responsibility in making decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that might endanger the public or the environment.

> This code applies to our project as it has a lot to do with the parts required for the assembly of aircraft. Thus, our project deals with the welfare of the public. We strive to minimize the future risk of our project.

3. To be honest and realistic in stating claims or estimates based on available data.

> As stated earlier, our project has a potential effect on public welfare. Therefore stating realistic claims and data is absolutely required for our project since lives may be at stake.

6. To maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations.

> The project is to explore NFC technology and apply it to the workspace. Thus it is our duty to obtain the correct information about this technology and deliver that to the people who would be using our end product.

7. To seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others.

> Our main goal is to satisfy our client and bring forth the deliverables that our client is expecting from us. Thus it is important for us to accept criticism of technical work, and credit properly the contributions of others who helped us throughout the project, so that we may improve our design and ideas.

9. To avoid injuring others, their property, reputation, or employment by false or malicious action

> Our project involves a client; therefore we avoided any action that would harm the reputation of our client. Also since our end product will be used by the employees of our client, it is absolutely important that our project is risk free.

Appendix

A. Requirements & Verifications

A.1 Part Tracking

Requirements	Verifications
NFC Badge	**NFC Badge**
1. Must be able to store and convey user credentials and employee identification number.	
a) Have 1KB of storage space	a) Store exactly 1KB of data on the tag and use the smartphone to scan the tag again and ensure that 1KB of data has been received.
b) Ability to be accessed in the range of 4cm ±2cm's	b) Using a caliper accurate to 0.1cm, place the tag exactly 4cm's away from the smartphone and scan the tag. Repeat the procedure at a distance of 2 cm's and 6cm's to verify that smartphone can still read the tag.
c) Have a max error percentage of 0.01% when being read for the data on a particular block	c) Read an arbitrary block that holds placeholder data for 120 times with the NFC shield and make sure that the data is read correctly every time.
Sensor Bundle	**Sensor Bundle**
1. Has to be able to measure temperature and acceleration and convey that information to the Arduino block.	
a) Capable of measuring temperatures between -40 to 120 (± 0.1 °C) °C.	a) With a pre-calibrated mercury based thermostat that is accurate to 1°C measure the room temperature. Compare that with the output of the digital temperature sensor bundle to verify is tolerance is in the given range.

b) Capable of measuring accelerations and tup to ±3G's (±0.1G).	b) Conduct a free-fall tests to with a pre-calibrated accelerometer that is accurate to ±0.1 G and determine the exact value of 1G.Make sure that the tolerance is ± 0.1G.
c) Both sensors must be able to function with 5V(±0.1) supplied from the Arduino	c) Power both sensors with Arduino's 5V pin and make sure that they are fully functional by running the code for Case Study 1.
d) Battery has to be capable of putting out 9 ±0.1 Volts and 50 ± 10 mA.	d) Using a multi-meter which is accurate 0.015 mV, measure the output of the power supply.
NFC Tag	**NFC Tag**
1. Must be able to store and convey part serial number and QA check results.	
a) Have 1KB of storage space	a) Store exactly 1KB of data on the tag and use the smartphone to scan the tag again and ensure that 1KB of data has been received.
b) Ability to be accessed in the range of 4cm ±2cm's	b) Using a caliper accurate to 0.1cm, place the tag exactly 4cm's away from the smartphone and scan the tag. Repeat the procedure at a distance of 2 cm's and 6cm's to verify that smartphone
c) Have a max error percentage of 0.01% when being read for the data on a particular block	c) Read an arbitrary block that holds placeholder data for 120 times with the NFC shield and make sure that the data is read correctly every time.
SD Card	**SD Card**
1. Must be able to store temperature data accurately	
a) Have 8GB of storage space	a) Store exactly 8GB of data on the card and use either a PC or a smartphone to make sure that all data is actually stored.
b) Have an error rate of less than 0.01% when receiving data from the sensor bundle	b) Store 120 arbitrary data points onto the SD card and make sure that there is no error to

	have 95% confidence in the said error rate.
LCD Screen	**LCD Screen**
1. Must be able display the required prompts without the need of an external power supply	
a) Has to function with 5V(± 0.1) supplied from the Arduino	a) Power up the LCD Screen from Arduino's 5V pin and run the Case Study 1 code to ensure full functionality

A.2 Heat Treatment

Requirements	Verifications
NFC Badge	**NFC Badge**
1. Must be able to store and convey user credentials and employee identification number.	
a) Have 1KB of storage space	a) Store exactly 1KB of data on the tag and use the smartphone to scan the tag again and ensure that 1KB of data has been received.
b) Ability to be accessed in the range of 4cm ±2cm's	b) Using a caliper accurate to 0.1cm, place the tag exactly 4cm's away from the smartphone and scan the tag. Repeat the procedure at a distance of 2 cm's and 6cm's to verify that smartphone can still read the tag.
c) Have a max error percentage of 0.01% when being read for the data on a particular block.	c) Read an arbitrary block that holds placeholder data for 120 times with the NFC shield and make sure that the data is read correctly every time.
Treatment Circuit	**Treatment Circuit**
1. Has to be able to simulate a heat treatment facility in a smaller scale, by measuring temperatures between 0 and 50 °C.	

a) **Thermistor** has to be able to detect a temperature range 0 to 50 ± 0.5 °C with an maximum error of 2% (± 1 °C)	a) With a pre-calibrated mercury based thermostat that is accurate to 1 °C measure the room temperature. Compare the recorded value with the measured value given by the test circuit in *Figure 2*.
b) **Op-amp** should have a maximum input current of 1nA (±0.1 nA)	b) Using a multi-meter which is accurate to 0.015 mV, measure the input voltage of the op-amp. Determine the input resistance using the same multi-meter. Use Ohm's Law to determine the input current.
d) Power Supply has to be capable of putting out 12 ±0.4 volts.	d) Using a multi-meter which is accurate 0.015 mV, measure the output of the power supply.
Arduino Uno R3	**Arduino Uno R3**
1. Has to be able to store and transmit data with high data integrity and a minimum processing rate of data.	
a) Must have a maximum error probability of 0.012%, while transmitting data wirelessly.	a) Send 90 packets to the Cosm storage and compare the received data with the original data, and calculate the rate of errors. Make sure it is under 0.012%.
2. Has to function without being dependant on the USB connection to a PC.	
a) Must be able to completely function with a 9V adaptor.	a) Plug the said adaptor to a wall outlet, and run codes for both case studies and verify that the codes have full functionality.
Cosm Cloud Storage	**Cosm Cloud Storage**
1. Has to be able to store temperature logs of multiple tags accurately.	
a) Must have a maximum error probability of 0.012%, while transmitting data wirelessly.	a) Send 90 packets to the Cosm storage and compare the received data with the original data, and calculate the rate of errors. Make sure

	it is under 0.012%.
b) Must be able to support at least 20 data streams.	b) Using a specific Arduino sketch, create 20 data streams with the help of a for loop. Store placeholder data to make sure Cosm is able to support all 20 streams.
NFC Tag	**NFC Tag**
1. Must be able to store and convey part serial number and QA check results.	
a) Have 1KB of storage space	a) Store exactly 1KB of data on the tag and use the smartphone to scan the tag again and ensure that 1KB of data has been received.
b) Ability to be accessed in the range of 4cm ±2cm's	b) Using a caliper accurate to 0.1cm, place the tag exactly 4cm's away from the smartphone and scan the tag. Repeat the procedure at a distance of 2 cm's and 6cm's to verify that smartphone can still read the tag.
LCD Screen	**LCD Screen**
1. Must be able display the required prompts without the need of an external power supply	
a) Has to function with 5V(± 0.1) supplied from the Arduino	a) Power up the LCD Screen from Arduino's 5V pin and run the Case Study 1 code to ensure full functionality

B. Cost Analysis

B.1 Labor

Name	Hourly Rate	Total Hours Invested	Total = HR x Hours Invested x 2.5
Alper O. Olcay	$ 40.00	300	$ 30,000
Jinjoo Nam	$40.00	300	$ 30,000
Vigneshwar Karthikeyan	$ 40.00	300	$ 30,000
Total		900	**$90,000**

B.2 Parts

Parts	Quantity	Unit Cost	Part Number	Cost
Arduino Uno R3	1	$ 29.95	Uno R3	$ 29.95
Arduino Uno R3 Starter Pack	2	$ 65.00	Uno R3	$ 130.00
NFC Tags	3	$ 10.00	MiFare Classic Tag Assortment	$ 30.00
Digital Accelerometer	2	$ 5.94	Analog Devices ADXL312	$ 11.88
Digital Temperature Sensor	2	$ 11.95	Freescale MPL115A2	$ 23.90
Hot Plate	1	$ 15.95		$ 15.95
Batteries	10	$ 1.29	Energizer 9V	$ 12.90
RCL Components	-	$ 10.00	-	$10.00
Thermistor	4	$0.66	Vishay Comp. NTCLE203E3	$2.64
Op-Amp	4	$5.36	Texas Ins. LF356	$21.44
Arduino NFC Shield	2	$39.95	Adafruit PN532	$79.90
Arduino WIFI Shield	2	$84.95	Arduino DEV-11287	$169.90

Arduino SD Card Shield	2	$14.95	DEV-09802	$29.90
2 Axis Accelerometer	2	$29.99	Memsic 2125	$59.98
Total				**$628.34**

B.3 Total

Section	**Cost**
Labor	$ 90,000
Parts	$ 628.34
Total	**$ 90,628.34**

C. Schematics & PCB Layouts

C.1 Part Tracking Schematics

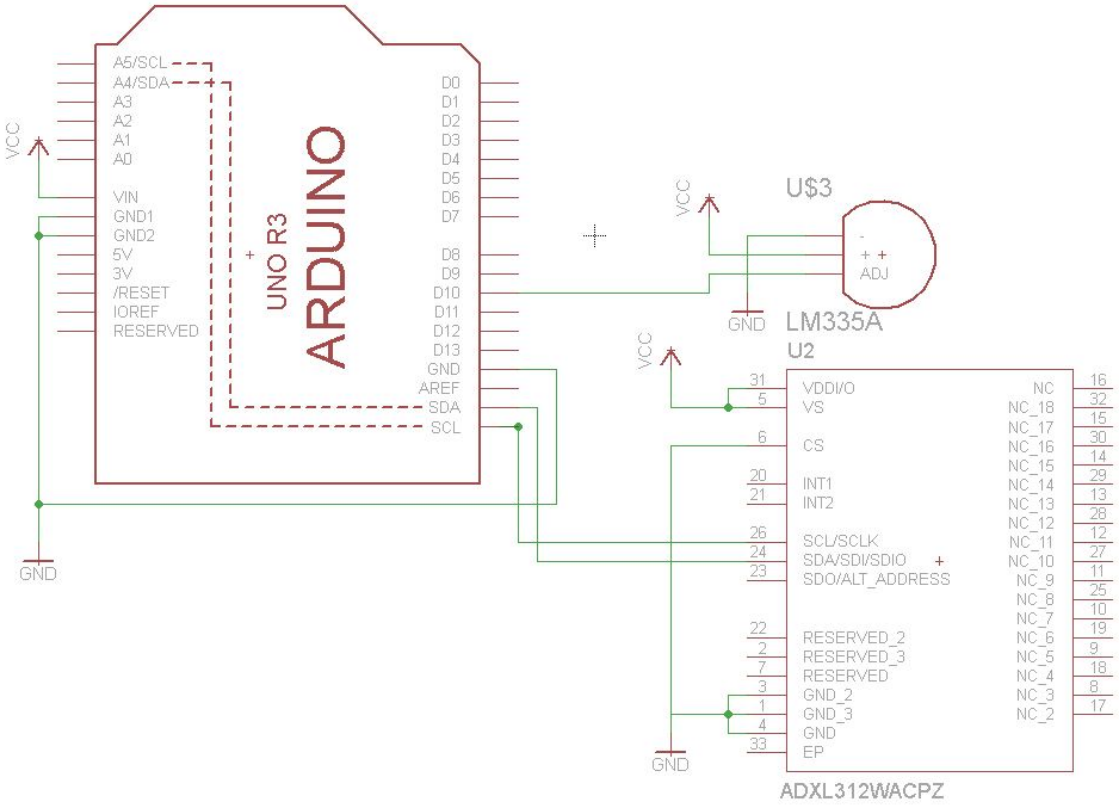

Figure 4: Part Tracking Schematic

C.2 Part Tracking Layout

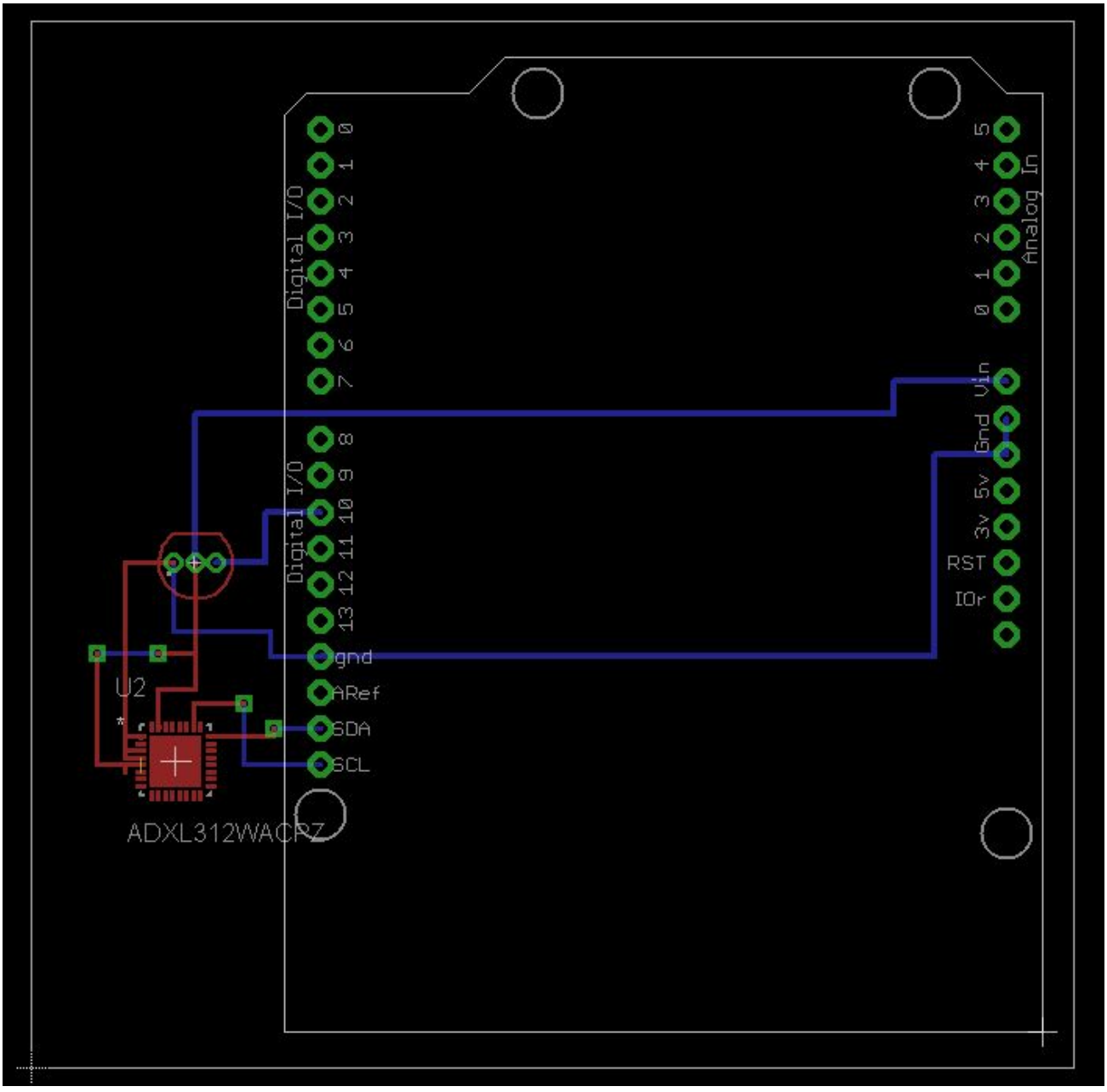

Figure 5: Part Tracking PCB Design

C.3 Heat Treatment Schematics

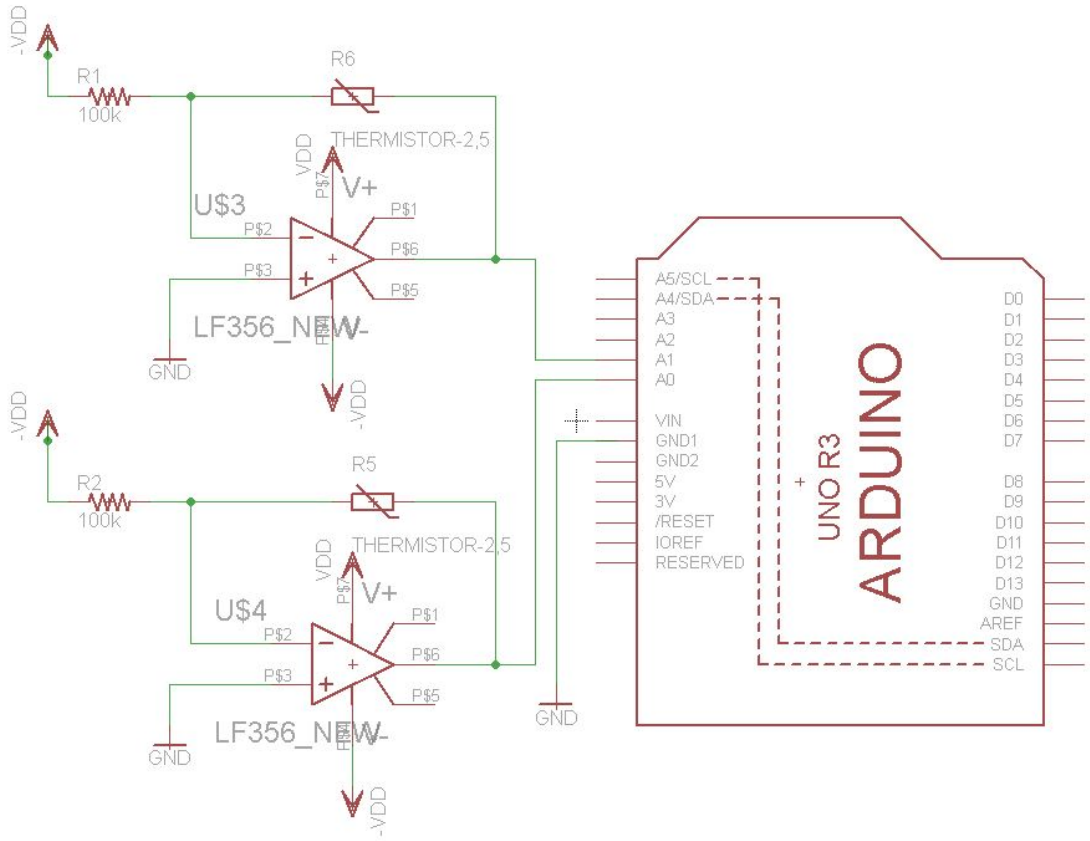

Figure 6:Heat Treatment Circuit Schematic

C.4 Heat Treatment Layout

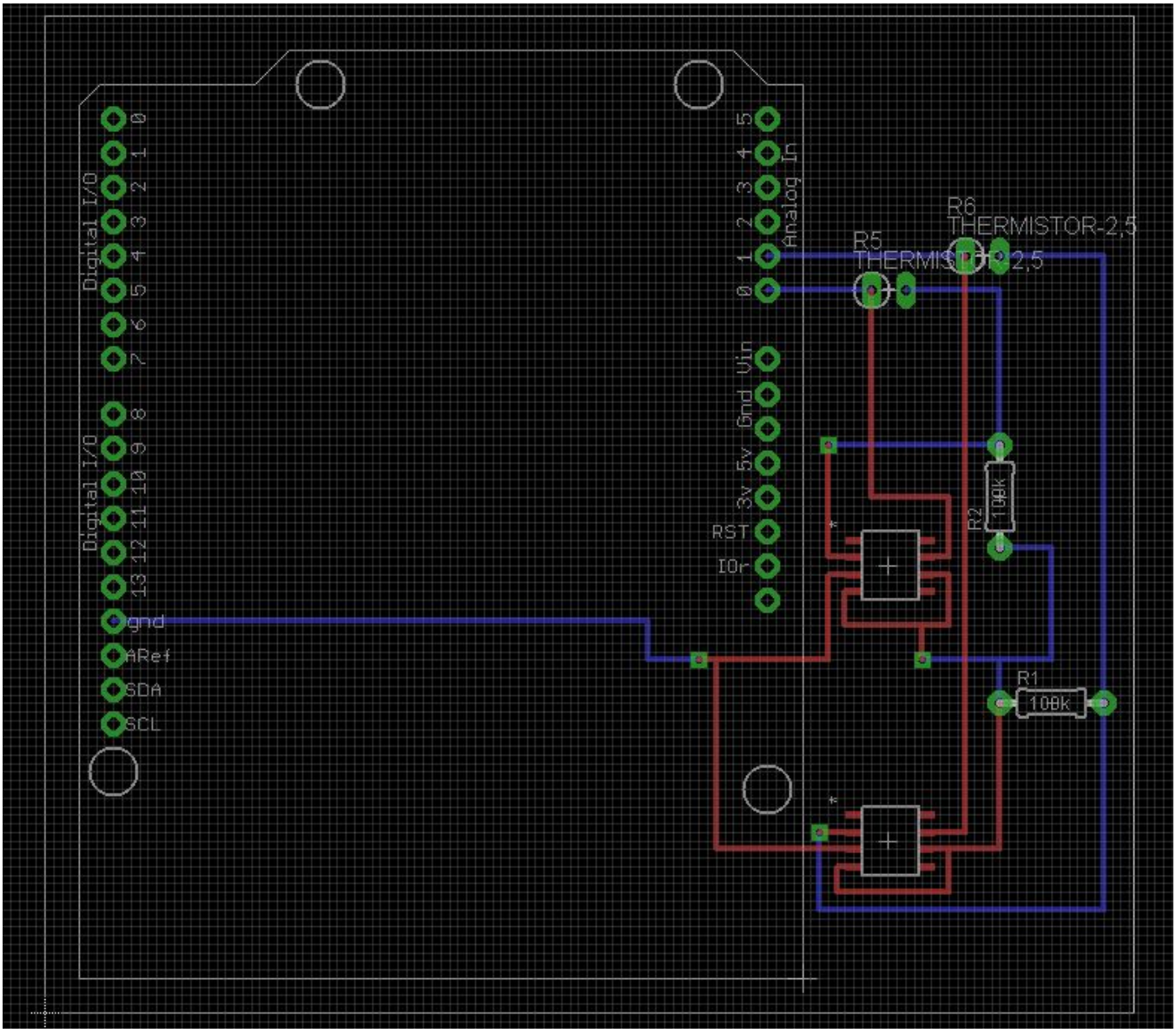

Figure 7: Heat Treatment PCB Design

D. Arduino Codes

D.1 Part Tracking Codes

```
/**************************************************************************/
/**************************************************************************/
#include <SPI.h>
#include <WiFi.h>
#include <math.h>
#include <Wire.h>
#include <Adafruit_NFCShield_I2C.h>
#include <Adafruit_MCP23017.h>
#include <Adafruit_RGBLCDShield.h>

#define IRQ   (2)
#define RESET (3)  // Not connected by default on the NFC Shield
#define RED 0x1
#define YELLOW 0x3
#define GREEN 0x2
#define TEAL 0x6
#define BLUE 0x4
#define VIOLET 0x5
#define WHITE 0x7

Adafruit_NFCShield_I2C nfc(IRQ, RESET);
Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield();

void setup(void) {
 Serial.begin(9600);
 Serial.println("Hello!");
 lcd.begin(16, 2);
 lcd.print("Boeing NFC");
 delay(3000);

 nfc.begin();

 uint32_t versiondata = nfc.getFirmwareVersion();
 if (! versiondata) {
  Serial.print("Didn't find PN53x board");
  while (1); // halt
 }
 // Got ok data, print it out!
```

```
  Serial.print("Found chip PN5"); Serial.println((versiondata>>24) & 0xFF, HEX);
  Serial.print("Firmware ver. "); Serial.print((versiondata>>16) & 0xFF, DEC);
  Serial.print('.'); Serial.println((versiondata>>8) & 0xFF, DEC);

  // configure board to read RFID tags
  nfc.SAMConfig();

  Serial.println("Waiting for an ISO14443A Card ...");
  LeReadBadge();
}

void loop(void)
{
 // Serial.println("In loop");
}
void LeReadBadge()
{

// Prompt User to Scan their badge.
// Call the LCD method to display the prompt
  LCD_Control(1);


// We want to read the data on an employee's badge and then see if they have proper access

// Read data on card
  uint8_t success;
  uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer to store the returned UID
  uint8_t uidLength;                        // Length of the UID (4 or 7 bytes depending on ISO14443A card type)

  // Wait for an ISO14443A type cards (Mifare, etc.).  When one is found
  // 'uid' will be populated with the UID, and uidLength will indicate
  // if the uid is 4 bytes (Mifare Classic) or 7 bytes (Mifare Ultralight)
  success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid, &uidLength);

 if (success) {
    // Display some basic information about the card
    Serial.println("Found an ISO14443A card");
    Serial.print("  UID Length: ");Serial.print(uidLength, DEC);Serial.println(" bytes");
    Serial.print("  UID Value: ");
    nfc.PrintHex(uid, uidLength);
    Serial.println("");
```

```
  if (uidLength == 4) {

    // We probably have a Mifare Classic card ...
    Serial.println("Seems to be a Mifare Classic card (4 byte UID)");

    // Now we need to try to authenticate it for read/write access
    // Try with the factory default KeyA: 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
    Serial.println("Trying to authenticate block 4 with default KEYA value");
    uint8_t keya[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };

          // Start with block 4 (the first block of sector 1) since sector 0
          // contains the manufacturer data and it's probably better just
          // to leave it alone unless you know what you're doing
    success = nfc.mifareclassic_AuthenticateBlock(uid, uidLength, 4, 0, keya);

    if (success)  {

      Serial.println("Sector 1 (Blocks 4..7) has been authenticated");
      // Check if the tag is indeed an Employee Tag
      uint8_t data1[16] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
      uint8_t data2[16] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
      Serial.println("Reading data from block 4: ");
      nfc.mifareclassic_ReadDataBlock(4, data1);
       // If block 4 holds a value of {0,0,0,1) it is an employee badge, otherwise {0,0,0,0} is a part tag
        if (success) {
          // Data seems to have been read ... spit it out
          Serial.println("Reading Block 4:");
          nfc.PrintHexChar(data1, 16);
          Serial.println("");
              int d1 = data1[15];
            if(d1 == 1){
//                Serial.println("Correct Type of Badge");
//                Serial.print("d1: "); Serial.println(d1);
              nfc.mifareclassic_ReadDataBlock(5, data2);// Read block 2 of the badge information
              int d2 = data2[15];
              Serial.print("d2: "); Serial.println(d2);
                 if(d2 == 1){
                     //uint8_t ID;
                     //nfc.mifareclassic_ReadDataBlock(6, ID);
                     LeReadTag();
                 }
```

```
                else {
                    LeReadBadge(); // failure
                }

            }
            else if(d1 == 0) {
                // Wrong tpye of tag
               Serial.println("Wrong type of tag, scan again");
               LeReadBadge();

            }

        }
         // Wait a bit before reading the card again
       delay(1000);
      }
      else {
       Serial.println("Ooops ... unable to read the requested block.  Try another key?");
      }
 }
 }
void LeReadTag()
{

 LCD_Control(2);  // "Scan part Tag"

 // Read data on card
  uint8_t success;
  uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer to store the returned UID
  uint8_t uidLength;                        // Length of the UID (4 or 7 bytes depending on ISO14443A card type)

  // Wait for an ISO14443A type cards (Mifare, etc.).  When one is found
  // 'uid' will be populated with the UID, and uidLength will indicate
  // if the uid is 4 bytes (Mifare Classic) or 7 bytes (Mifare Ultralight)
  success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid, &uidLength);

 if (success) {
   // Display some basic information about the card
   Serial.println("Found an ISO14443A card");
   Serial.print("  UID Length: ");Serial.print(uidLength, DEC);Serial.println(" bytes");
   Serial.print("  UID Value: ");
   nfc.PrintHex(uid, uidLength);
```

```
Serial.println("");

if (uidLength == 4) {

 // We probably have a Mifare Classic card ...
 Serial.println("Seems to be a Mifare Classic card (4 byte UID)");

 // Now we need to try to authenticate it for read/write access
 // Try with the factory default KeyA: 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
 Serial.println("Trying to authenticate block 4 with default KEYA value");
 uint8_t keya[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };

        // Start with block 4 (the first block of sector 1) since sector 0
        // contains the manufacturer data and it's probably better just
        // to leave it alone unless you know what you're doing
 success = nfc.mifareclassic_AuthenticateBlock(uid, uidLength, 4, 0, keya);

 if (success)  {

   Serial.println("Sector 1 (Blocks 4..7) has been authenticated");
   // Check if the tag is indeed an Employee Tag
   uint8_t data1[16] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
   uint8_t data2[16] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
   Serial.println("Reading data from block 4: ");
   nfc.mifareclassic_ReadDataBlock(4, data1);
    // If block 4 holds a value of {0,0,0,1) it is an employee badge, otherwise {0,0,0,0} is a part tag
     if (success) {
        // Data seems to have been read ... spit it out
        Serial.println("Reading Block 4:");
        nfc.PrintHexChar(data1, 16);
            int d1 = data1[15];
          if(d1 == 1){
             Serial.println("Wrong Type of Card");
             LeReadTag();
          }
          else if(d1 == 0) {
              // Correct type of tag
             Serial.println("Correct type of tag");


              uint8_t Success1[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};
              nfc.mifareclassic_WriteDataBlock(6, Success1);
```

```
                //nfc.mifareclassic_WriteDataBlock(7, EmployeeID);
                End_Process();
               }
           }
          // Wait a bit before reading the card again
        delay(1000);
      }
      else {
        Serial.println("Ooops ... unable to read the requested block.  Try another key?");
      }
}
}
}
void End_Process () {

 LCD_Control(5);
 delay(10000);
 End_Process(); // Loop until the arduino has been reset for another part for the furnace
}
void LCD_Control(int Choice) {
 // Depending on the condition, print different message for the user

 // Clear the LCD screen before new message is displayed
 lcd.clear();
 lcd.setCursor(0,1);
 lcd.clear();
 lcd.setCursor(0,0);

 if(Choice == 1){
  // Scan employee badge prompt
 lcd.print("Scan Badge");
 delay(1000);

 }
 else if(Choice == 2) {
  lcd.print("Scan Part Tag");
  delay(1000);

 }
 else if(Choice == 3)  {
  lcd.print("Wrong Card Type");
  delay(1000);
```

```
 }
 else if(Choice == 4)  {
  lcd.print("Invalid Access");
  delay(1000);
 }
 else if(Choice == 5)  {
  lcd.print("QA Complete");
  lcd.setCursor(0,1);
  lcd.print("Reset Arduino");

 }
}
#include <SD.h>
#include <Wire.h>
#include <Adafruit_MPL115A2.h>

Adafruit_MPL115A2 mpl115a2;

// On the Ethernet Shield, CS is pin 4. Note that even if it's not
// used as the CS pin, the hardware CS pin (10 on most Arduino boards,
// 53 on the Mega) must be left as an output or the SD library
// functions will not work.
const int chipSelect = 4;
const int xPin = 2;                  // X output of the accelerometer
const int yPin = 3;                  // Y output of the accelerometer


void setup()
{
 // Open serial communications and wait for port to open:
  Serial.begin(9600);
  mpl115a2.begin();
   while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }


  Serial.print("Initializing SD card...");
  // make sure that the default chip select pin is set to
  // output, even if you don't use it:
  pinMode(10, OUTPUT);
```

```
 pinMode(xPin, INPUT);
 pinMode(yPin, INPUT);

 // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect)) {
   Serial.println("Card failed, or not present");
   digitalWrite(6, HIGH);
   return;
   // don't do anything more:
 }
 Serial.println("card initialized.");
}

void loop()
{
 if (SD.begin(chipSelect)){
   digitalWrite(6, LOW);
 }

 // make a string for assembling the data to log:
 File dataFile = SD.open("datalou.txt", FILE_WRITE);
 String dataString = "";
 int temperatureC = 0;
 // variables to read the pulse widths:
 int pulseX, pulseY;
 // variables to contain the resulting accelerations
 int accelerationX, accelerationY;
   // read pulse from x- and y-axes:
 pulseX = pulseIn(xPin,HIGH);
 pulseY = pulseIn(yPin,HIGH);
 accelerationX = ((pulseX / 10) - 500) * 8;
 accelerationY = ((pulseY / 10) - 500) * 8;
 temperatureC = mpl115a2.getTemperature();
 if(temperatureC <= 20 || temperatureC > 25)
 {
 dataString += "Temp:";
 dataString += String(temperatureC);
 dataFile.println(dataString);
 Serial.println(dataString);
 }
 if (accelerationX<=-500 || accelerationX >= 500)
 {
```

```
 dataString = "";
 dataString += "X axis acceleration:";
 dataString += String(accelerationX);
 dataFile.println(dataString);
 Serial.println(dataString);
 }
 if (accelerationY<=-500 || accelerationY>= 500)
 {
 dataString = "";
 dataString += "Y axis acceleration:";
 dataString += String(accelerationY);
 dataFile.println(dataString);
 Serial.println(dataString);

 }

   dataFile.close();
   // print to the serial port too:


  // if the file isn't open, pop up an error:

}
```

D.2 Heat Treatment Codes

```
/*
Senior Design ECE 445
Boeing NFC
Team 41
 Heat Treatment Case Study
*/
#include <SPI.h>
#include <WiFi.h>
#include <math.h>
#include <Wire.h>
#include <Adafruit_NFCShield_I2C.h>
#include <Adafruit_MCP23017.h>
#include <Adafruit_RGBLCDShield.h>

#define IRQ   (2)
#define RESET (3)  // Not connected by default on the NFC Shield

Adafruit_NFCShield_I2C nfc(IRQ, RESET);
Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield();


#define RED 0x1
#define YELLOW 0x3
#define GREEN 0x2
#define TEAL 0x6
#define BLUE 0x4
#define VIOLET 0x5
#define WHITE 0x7

int Part_Serial_Num = 0; // Initialize the part serial number that will contain serial number from the part
tag


#define APIKEY         "LzE87MgfHbVuC5NopeUs2L_-yY2SAKxra1VCYW9kc0R3az0g" // replace your
pachube api key here
#define FEEDID         126515               // replace your feed ID
#define USERAGENT      "Osman"     // user agent is the project name
```

```
char ssid[] = "Galaxy_S_III_1851";      //  your network SSID (name)
char pass[] = "Blahblah";   // your network password

int status = WL_IDLE_STATUS;

// initialize the library instance:
WiFiClient client;
// if you don't want to use DNS (and reduce your sketch size)
// use the numeric IP instead of the name for the server:
IPAddress server(216,52,233,121);      // numeric IP for api.pachube.com
//char server[] = "api.pachube.com";   // name address for pachube API

unsigned long lastConnectionTime = 0;         // last time you connected to the server, in milliseconds
boolean lastConnected = false;                // state of the connection last time through the main loop
const unsigned long postingInterval = 10*1000; //delay between updates to pachube.com

void setup() {
 //Initialize serial and wait for port to open:
 Serial.begin(9600);
 //while (!Serial) {
  ; // wait for serial port to connect. Needed for Leonardo only
 //}
 Serial.println("Hello!");
 // Begin the LCD display
 lcd.begin(16, 2);
 lcd.print("Boeing NFC");


 // attempt to connect to Wifi network:
 while ( status != WL_CONNECTED) {
  Serial.print("Attempting to connect to SSID: ");
  Serial.println(ssid);
  // Connect to WPA/WPA2 network. Change this line if using open or WEP network:
  status = WiFi.begin(ssid, pass);

  // wait 10 seconds for connection:
  delay(10000);
 }
 // you're connected now, so print out the status:
 printWifiStatus();
```

```
 nfc.begin();

 uint32_t versiondata = nfc.getFirmwareVersion();
 if (! versiondata) {
  Serial.print("Didn't find PN53x board");
  while (1); // halt
 }
 // Got ok data, print it out!
 Serial.print("Found chip PN5"); Serial.println((versiondata>>24) & 0xFF, HEX);
 Serial.print("Firmware ver. "); Serial.print((versiondata>>16) & 0xFF, DEC);
 Serial.print('.'); Serial.println((versiondata>>8) & 0xFF, DEC);

 // configure board to read RFID tags
 nfc.SAMConfig();

 Serial.println("Waiting for an ISO14443A Card ...");

//  delay(100);
//  Serial.println("Calling NFC Badge Method");
  Read_NFC_Badge();

// Call the part tag to be read before the process
  Read_PartTag();

}

int Read_NFC_Badge(  ) {

// Prompt User to Scan their badge.
// Call the LCD method to display the prompt
  LCD_Control(1);


// We want to read the data on an employee's badge and then see if they have proper access
// Read data on card
       // Check if the tag is indeed an Employee Tag
       uint8_t data1[16] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
       uint8_t data2[16] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
       Serial.println("Reading data from block 4: ");
       nfc.mifareclassic_ReadDataBlock(4, data1);
         // If block 4 holds a value of {0,0,0,1) it is an employee badge, otherwise {0,0,0,0} is a part tag
          if (success) {
```

```
            // Data seems to have been read ... spit it out
   //          Serial.println("Reading Block 4:");
            nfc.PrintHexChar(data1, 16);
            Serial.println("");
      int d1 = data1[15];
              if(d1 == 1){
//                  Serial.println("Correct Type of Badge");
//                  Serial.print("d1: "); Serial.println(d1);
                 nfc.mifareclassic_ReadDataBlock(5, data2);// Read block 2 of the badge information
                 int d2 = data2[15];
//                  Serial.print("d2: "); Serial.println(d2);
                   if(d2 == 1){
                    //    LCD_Control(10);
                       return 1; // indicate success of having write permission
                   }
                   else {
                       Read_NFC_Badge(); // failure
                       return 1;
                   }

              }
              else if(d1 == 0) {
                  // Wrong tpye of tag
                 Serial.println("Wrong type of tag, scan again");
                 Read_NFC_Badge();
                 return 1;

              }

        }
         // Wait a bit before reading the card again
       delay(1000);
      }
      else {
       Serial.println("Ooops ... unable to read the requested block.  Try another key?");
      }
    }

  }
 }
void Read_PartTag () {
  // LCD "Scan part Tag"
```

```
  LCD_Control(2);
  Serial.println("Waiting for an ISO14443A Card ...");
// Read data on card
       // Check if the tag is indeed an Employee Tag
       uint8_t data1[16] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
       uint8_t data2[16] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
       Serial.println("Reading data from block 4: ");
       nfc.mifareclassic_ReadDataBlock(4, data1);
        // If block 4 holds a value of {0,0,0,1) it is an employee badge, otherwise {0,0,0,0} is a part tag
         if (success) {
            // Data seems to have been read ... spit it out
  //            Serial.println("Reading Block 4:");
            nfc.PrintHexChar(data1, 16);
            Serial.println("");
      int d1 = data1[15];
              if(d1 == 1){
                 Serial.println("Wrong Type of Card");
                 Read_PartTag();
              }
              else if(d1 == 0) {
                  // Correct type of tag
                 Serial.println("Correct type of tag");
                 nfc.mifareclassic_ReadDataBlock(5, data2);
                 int d2 = data2[15]*1+data2[14]*2+data2[13]*4+data2[12]*8;
                 Part_Serial_Num = d2;
//                  Serial.print("Part Serial Number: "); Serial.println(Part_Serial_Num);
              }

         }
          // Wait a bit before reading the card again
        delay(1000);
       }
       else {
        Serial.println("Ooops ... unable to read the requested block.  Try another key?");
       }
    }
  }
}
int Write_To_PartTag (int Success_of_process) {

  LCD_Control(2);  // "Scan part Tag"
```

```
// Read data on card
      // Check if the tag is indeed an Employee Tag
      uint8_t data1[16] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
      uint8_t data2[16] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
      Serial.println("Reading data from block 4: ");
      nfc.mifareclassic_ReadDataBlock(4, data1);
       // If block 4 holds a value of {0,0,0,1) it is an employee badge, otherwise {0,0,0,0} is a part tag
        if (success) {
          // Data seems to have been read ... spit it out
//            Serial.println("Reading Block 4:");
          nfc.PrintHexChar(data1, 16);
          Serial.println("");
     int d1 = data1[15];
            if(d1 == 1){
              Serial.println("Wrong Type of Card");
              Read_PartTag();
            }
            else if(d1 == 0) {
               // Correct type of tag
              Serial.println("Correct type of tag");

              if(Success_of_process == 1){  // Success of industrial process
                  uint8_t Success1[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};
                  nfc.mifareclassic_WriteDataBlock(3, Success1);
                   //       nfc.mifareclassic_WriteDataBlock(4, EmployeeID);
                  return 1;
              }
              else if(Success_of_process == 0){   // Failure of industrial process
                  uint8_t Success1[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
                  nfc.mifareclassic_WriteDataBlock(3, Success1);
                   //       nfc.mifareclassic_WriteDataBlock(4, EmployeeID);
                  return 1;
              }
            }

        }
         // Wait a bit before reading the card again
       delay(1000);
     }
     else {
       Serial.println("Ooops ... unable to read the requested block.  Try another key?");
     }
```

```
   }
  }
}

float TemperatureSensor (  ) {

  int Thermistorpin1 = A0;
//  int Thermistorpin2 = A1;
  float rawVoltage1 = analogRead(Thermistorpin1);     // value from sensor / .12 - Gives resistance value
of the thermistor
//  float rawVoltage2 = analogRead(Thermistorpin2);  // value from sensor / .12 - Gives resistance value
of the thermistor 2
  float Resistor1 = rawVoltage1 * (5.0/1023.0)/(12.0/98000.0);
//  float Resistor2 = rawVoltage2 * (5.0/1023.0)/(12.0/98000.0);
  float Temperature1 = 1.0/(.000251*log(Resistor1/98000.0)+1.0/298.0)-273.0-79.0;  // Calculating
Temperature from Rt
//  float Temperature2 = 1.0/(.000251*log(Resistor2/98000.0)+1.0/298.0)-273.0-56.0;  // Calculating
Temperature from Rt

  Serial.print("Thermistor 1: ");       // prints a label
  Serial.println(Temperature1);     // print the resistance value
//  Serial.print("Thermistor 2: ");       // prints a label
//  Serial.println(Temperature2);     // print the resistance value at 2

  return Temperature1;



}

int Temperature_Control (float Temperature1) {
 // Check to see if the maximum temperature has been reached by the furnace

  if (Temperature1 >= 50 && Temperature1 <= 80) {
    // Part has reached maximum temperature needed,
    // Call the LCD method to print out the stop command
    LCD_Control(8);  // "Turn off the heat"
    delay(10000);
    //
       return 1; // We want to return and write to the part tag for success or failure
  }
  else if(Temperature1 <= 50) {
```

```
  //
  return 2; // indicate that temperature is still 50 so continue logging data from sensors.
 }
 else if(Temperature1 >= 80) {
     // Temperature has reached too high a temperature, process has failed.
  LCD_Control(8);
  LCD_Control(6);
  return 3;
 }
}

void LCD_Control(int Choice) {
// Depending on the condition, print different message for the user

// Clear the LCD screen before new message is displayed
lcd.clear();
lcd.setCursor(0,1);
lcd.clear();
lcd.setCursor(0,0);

if(Choice == 1){
  // Scan employee badge prompt
 lcd.print("Scan Badge");
 delay(1000);

}
else if(Choice == 2) {
  lcd.print("Scan Part Tag");
  delay(1000);

}
else if(Choice == 3)  {
  lcd.print("Wrong Card Type");
  delay(3000);

}
else if(Choice == 4)  {
  lcd.print("Invalid Access");
  delay(3000);

}
else if(Choice == 5)  {
```

```
  lcd.print("Successful");
  delay(3000);

 }
 else if(Choice == 6)  {
  lcd.print("Process Failed");
  delay(1000);

 }
 else if(Choice == 7)  {
  lcd.print("Turn on Heat");
  delay(1000);

 }
 else if(Choice == 8)  {
  lcd.print("Turn off Heat");
  delay(1000);

 }
 else if(Choice == 9)  {
  lcd.print("Reset Arduino To");
  lcd.setCursor(0,1);
  lcd.print("Start New Part");
  delay(10000);

 }
 else if(Choice == 10)  {
  lcd.print("Data Logging");
  delay(1000);
 }
 }

void loop() {

 // Find temperature and check with the control method
  LCD_Control(10);  // LCD print "Data Logging"
  float Temperature1 = TemperatureSensor();    // Call method to get Temperature Value
  int Success = Temperature_Control(Temperature1);  // Call control method to find the range of the
temperature value, arbitrary value of Success of Badge read

  if(Success == 1) {
   // Continue to log data
```

```
 // if there's no net connection, but there was one last time
 // through the loop, then stop the client:
  if (!client.connected() && lastConnected) {
    Serial.println();
    Serial.println("disconnecting.");
    client.stop();
  }

  // if you're not connected, and ten seconds have passed since
  // your last connection, then connect again and send data:
  if(!client.connected() && (millis() - lastConnectionTime > postingInterval)) {
    sendData(Temperature1);
  }
  // store the state of the connection for next time through
  // the loop:
  lastConnected = client.connected();

 LCD_Control(5);
  // Write success of process
 Write_To_PartTag(Success);
 End_Process();

}
else if(Success == 2) {
   // we want to log the data to the server client
   // if there's no net connection, but there was one last time
   // through the loop, then stop the client:
  if (!client.connected() && lastConnected) {
    Serial.println();
    Serial.println("disconnecting.");
    client.stop();
  }

  // if you're not connected, and ten seconds have passed since
  // your last connection, then connect again and send data:
  if(!client.connected() && (millis() - lastConnectionTime > postingInterval)) {
    sendData(Temperature1);
  }
  // store the state of the connection for next time through
  // the loop:
  lastConnected = client.connected();
```

```
 }
 else if(Success == 3) {
  // Process fail, write that onto the part tag
  Success = 0;
  // if there's no net connection, but there was one last time
 // through the loop, then stop the client:
  if (!client.connected() && lastConnected) {
   Serial.println();
   Serial.println("disconnecting.");
   client.stop();
  }

  // if you're not connected, and ten seconds have passed since
  // your last connection, then connect again and send data:
  if(!client.connected() && (millis() - lastConnectionTime > postingInterval)) {
   sendData(Temperature1);
  }
  // store the state of the connection for next time through
  // the loop:
  lastConnected = client.connected();

  LCD_Control(6);
  delay(10000);
  Write_To_PartTag(Success);
  End_Process();
 }


}

// this method makes a HTTP connection to the server:
void sendData(int thisData) {
 // if there's a successful connection:
 if (client.connect(server, 80)) {
  Serial.println("connecting...");
  // send the HTTP PUT request:
  client.print("PUT /v2/feeds/");
  client.print(FEEDID);
  client.println(".csv HTTP/1.1");
  client.println("Host: api.pachube.com");
  client.print("X-ApiKey: ");
  client.println(APIKEY);
```

```
    client.print("User-Agent: ");
    client.println(USERAGENT);
    client.print("Content-Length: ");

    // calculate the length of the sensor reading in bytes:
    // 8 bytes for "sensor1," + number of digits of the data:
    int thisLength = 8 + getLength(thisData);
    client.println(thisLength);

    // last pieces of the HTTP PUT request:
    client.println("Content-Type: text/csv");
    client.println("Connection: close");
    client.println();

    // here's the actual content of the PUT request:

    // Change the Part_Serial_Num to a char string so that it can be loaded to the database
    if(Part_Serial_Num == 5) {;
      client.print("Part5,");
      client.println(thisData);
    }
    else if(Part_Serial_Num == 10) {
      client.print("Part10,");
      client.println(thisData);
    }
  }
  else {
    // if you couldn't make a connection:
    Serial.println("connection failed");
    Serial.println();
    Serial.println("disconnecting.");
    client.stop();
  }
   // note the time that the connection was made or attempted:
  lastConnectionTime = millis();
}

// This method calculates the number of digits in the
// sensor reading.  Since each digit of the ASCII decimal
// representation is a byte, the number of digits equals
// the number of bytes:
```

```
int getLength(int someValue) {
  // there's at least one byte:
  int digits = 1;
  // continually divide the value by ten,
  // adding one to the digit count for each
  // time you divide, until you're at 0:
  int dividend = someValue /10;
  while (dividend > 0) {
    dividend = dividend /10;
    digits++;
  }
  // return the number of digits:
  return digits;
}

void printWifiStatus() {
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your WiFi shield's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}

void End_Process ( ) {

  LCD_Control(9);
  End_Process(); // Loop until the arduino has been reset for another part for the furnace

}
```

E. References

[1] “TEXAS INSTRUMENTS: Low-Power, Digital Temperature Sensor with Two-Wire Interface in WCSP” DataSheet[Online]. Available: http://www.ti.com/lit/ds/symlink/tmp103.pdf

[2]“TEXAS INSTRUMENTS: JFET Input Operational Amplifiers” DataSheet[Online]. Available: http://www.ti.com/lit/ds/symlink/lf356.pdf

[3]“TEXAS INSTRUMENTS: Precision Temperature Sensors” DataSheet[Online]. Available: http://www.ti.com/lit/ds/symlink/lm335.pdf

[4] “Boeing RFID Applications: Confidential”

[5] “ANALOG DEVICES: Digital Accelerometer ADXL312” DataSheet[Online]. Available: http://www.analog.com/static/imported-files/data_sheets/ADXL312.pdf

[6] “VISHAY: NTC Thermistors, Radial Leaded, Accuracy Line” DataSheet[Online]. Avaliable: http://www.vishay.com/docs/29048/ntcle203.pdf

[7] IEEE Code of Ethics[Online]. Available: http://www.ieee.org/about/corporate/governance/p7-8.html

[8] “PARALLAX: Memsic 2125 Dual-Axis Accelerometer(#28017)” Datasheet[Online]. Available: http://www.parallax.com/Portals/0/Downloads/docs/prod/sens/28017-Memsic2Axis-v2.0.pdf

[9]"Wireless Reliability: Rethinking 802.11 Packet Loss."University of Notre Dame. David C. Salyers, Aaron Striegel, Christian Poellabauer, n.d.[Online]. Avaliable: http://www.cse.nd.edu/Reports/2007/TR-2007-06.pdf

[10]”FREESCALE SEMICONDUCTOR: Miniature I2C Digital Barometer & Temperature Sensor” Datasheet[Online]. Available: http://www.adafruit.com/datasheets/MPL115A2.pdf