Lightning Strike Detector

Michigan State University
Senior Design-ECE 480
April 29, 2015

Project Sponsors:
Greg Hoshal
Instrumented Sensor Technology

Faculty Facilitator:
Dr. Robert McGough

Team Members:
Justin Bauer
Matt Clary
Adam McHale
DeAndre Dawson
Zongheng Pu
Acknowledgments

**Greg Hoshal:** Special thanks to the team’s corporate sponsor Instrumented Sensor Technology and especially Greg Hoshal. Greg Hoshal provided great insight toward project goals and expectations. He also provided the team with valuable equipment that was used to test and design the Lightning Strike Detector. We appreciate the extra effort given to allow a successful project.

**Dr. S. Ratnajeevan H. Hoole:** Special thanks to Dr. Hoole as the team’s faculty advisor. Dr. Hoole worked with the team to perfect all reports and presentations. He also provided insight toward the characteristics of lightning and different approaches on how to detect it. We appreciate the opinions and expertise shown during meetings.

**Greg Mulder and Brian Wright:** Special thanks the Greg Mulder and Brian Wright. They both helped the design team with using certain equipment and especially with designing the device’s enclosure. They also went beyond that to offer some suggestions and opinions on how to do things. We appreciate all the work you both did to help finish our project.

**Executive Summary**

The purpose of the lightning strike detector project is to develop a detection system for one of the most common issues with data loss on business computers. This data loss is due to the electromagnetic fields and secondary currents caused by lightning strikes. The team will build a circuit using a series of different sensors and coils to determine when a lightning induced magnetic field is present. Then, the team will use a microcontroller and an LCD to count how often such surges occur.
# Table of Contents

Chapter 1: Introducing the Problem
- 1.1 Introduction ........................................... 3
- 1.2 Background ........................................... 3
- 1.3 Current Designs ........................................... 4
- 1.4 The Approach ........................................... 5

Chapter 2: Exploring a Solution and Selecting an Approach
- 2.1 Exploring the Solution ....................................... 6
- 2.2 Selecting a Solution ........................................... 8
- 2.3 Budget ........................................... 9
- 2.4 Gantt Chart ........................................... 10

Chapter 3: Technical Work Performed
- 3.1 Hardware
  - 3.1.1 Sensing and Amplifier Circuit ............................ 11
  - 3.1.2 One-Shot Multivibrator .................................... 12
  - 3.1.3 Arduino-LCD Interface (Hardware) ....................... 14
  - 3.1.4 Enclosure ........................................... 19
- 3.2 Software
  - 3.2.1 Arduino-LCD Interface (Software) ....................... 20

Chapter 4: Test Data and Proof of Design
- 4.1 Test Procedures ........................................... 27
- 4.2 Project Functionality ........................................... 30
- 4.3 Sustainability ........................................... 31

Chapter 5: Summary, Final Costs and Future Improvements
- 5.1 Summary ........................................... 31
- 5.2 Final Costs ........................................... 32
- 5.3 Future Improvements ........................................... 34

Appendix 1 ........................................... 35
Appendix 2 ........................................... 40
Appendix 3 ........................................... 41
Chapter 1: Introducing the Problem

1.1 Introduction

Lightning strikes are a powerful natural phenomenon that can cause catastrophic damage to electrical devices within the vicinity of the strike if they are not properly protected. The average lightning strike carries an average current of 30 kA and lasts only the span of 60 to 200 microseconds. Most structures are protected from lightning strikes by utilizing special grounding systems, designed to route large amounts of electrical current to ground through a safe, controlled route. By routing the current away from the parts of structure susceptible to high currents the structure can be protected from any major physical damage due to lightning.

At particular facilities it is beneficial to know, and keep count of, the occurrence of lightning strikes. This knowledge can be used to perform maintenance on the grounding systems of such structures, as well as for the inspection of specific parts of the structure (computers, automated control systems, etc.). Although grounding systems provide a degree of protection to the structure itself, the EMP given off by the lightning strike can cause errors in any electrical systems that are operating at the time of the strike. The team's goal in this project was to design a lightning strike detector that can identify and record lightning strikes that occur at a particular facility, thus ensuring the maintenance of the facility can take the proper steps to fix any issues that may have been caused by the strike.

1.2 Background

Lightning is an electrostatic discharge that occurs between electrically charged regions of a cloud and the earth. Occurring during electrical storms, lightning is due to the buildup of electrical charges within the cloud and on the earth below. The top of a storm cloud typically accumulates a large positive charge, while the base of the cloud builds up a large negative charge. A region of positive charge also follows the cloud on the earth below, and usually tends to concentrate on tall objects. A lightning strike begins when a channel of negative charges begins to make its way to the ground. At the same time, currents of positive charges begin to move upward from the ground. When these two charges eventually meet, it causes a large electric current to flow up to the cloud. Positive charges of the Earth tend to gravitate toward
tall structures so they can be close to the negative charges of the storm cloud. As a result, many large buildings can be heavily damaged due to a lightning strike. In order to protect these structures, precautionary devices known as lightning rods are often installed to reduce the damage of a lightning strike.

A lightning rod is a metallic object mounted on top of a structure designed to provide a safe route to ground the current produced by a lightning strike. The rod protects the structure from any significant damage that could potentially be caused by lightning. A typical lightning strike carries thousands of amps of current. The result of lightning striking an unprotected structure can cause serious damage. However, lightning rods are not 100 percent effective, as many computer systems and electronics are still susceptible to significant damage from lightning. This damage is due to the electromagnetic field that is produced by the strike, as well as secondary currents that can be induced in electrical wiring. The damage could potentially cause a variety of errors in any computer or electrical systems that are running during the time of the strike. Because of this, it is desirable to know when lightning has struck a certain structure. This allows one to check any computer systems or electronics to see if any errors occurred during that time.

1.3 Current Designs

Currently, a variety of lightning strike detectors exist on the market for both professional and personal use. Most of these detectors are based around circuitry designed to pick up the RF electromagnetic signal produced by the lightning strike. However, this alone is not a very effective tactic in detecting lightning, as RF signals from outside sources can trigger false alarms. Due to this, some lightning strike detectors use electromagnetic field detection in combination with circuitry designed to detect the light pulse given out by lightning. Since few sources other than lightning produce both a light pulse as well as an electromagnetic pulse, this method has been found to be effective in detecting a lightning strike. These circuits are often designed to detect lightning from a distance. Because of this, most of these devices are not well suited to this project, as the goal is to detect only when lightning strikes a certain building or structure.
In order to design a device to better suit to meet the goals outlined by the customer the team has made a variety of changes to the devices already on the market. As mentioned before, most lightning strike detectors use the EMP radiated from lightning when a strike occurs. EMP energy can be transferred in a variety of forms, including electric fields, magnetic fields, electromagnetic radiation, and electrical conduction. This presents a few different problems in designing the final product. First, while energy transmitted through electrical fields, magnetic fields, and electrical conduction act only over relatively short distances, electromagnetic radiation acts over a long range. This means lightning that occurs far away from user’s location can trigger the device. The second issue with this method of detection is that it is very susceptible to outside interference from anything that radiates energy in any of the forms mentioned above can trigger the device in addition to lightning.

1.4 The Approach

To address these issues the team sought a different way to detect a lightning strike within a very limited range. The solution chosen by the team was to design a device to detect the magnetic field radiated from the grounding rod of a building as it is struck by lightning. Because lightning can produce anywhere from 5-100 kA of current, the magnetic field radiated from a grounding rod as it is struck by lightning is very strong, as well as brief. By shielding the device from the other components of the EMP given off by lightning and isolating the magnetic field, any outside interference could be greatly reduced. Due to the rate at which the magnetic field dissipates through air the device can also be limited to detecting a strike within a few feet of a desired location while ignoring strikes that occur at further distances away. This aspect of the design was very important to the team because it separated what the team wanted to do from designs that are currently on the market.
2.1 Exploring the Solution

The Lightning Strike Detector project has several different functions involved which require different subsystems. These subsystems can be seen in the FAST diagram shown in figure 1. The main functions are to detect a lightning strike, record the time of a lightning strike, count the amount of lightning strikes, and withstand the weather and to perform over a long period of time. Each of these main functions required specific detail so that they could be implemented smoothly within the design.

The main function of the Lightning Strike Detector is the ability to detect a lightning strike. Lightning has many characteristics that can be sensed with different circuit elements. Some of these characteristics include electrostatic discharge and magnetic field due to current flowing through a grounding rod. Sensing electrostatic discharge would require the device to have an antenna. This antenna would act as the sensing circuit element for the device. Sensing a magnetic field could be done with a variety of different sensors. Sensors such as magneto
resistors, inductors and Hall Effect sensors could be used. The main issue to focus on in this subsystem is to ensure that only the lightning strike is being sensed by the device. External noise, such as radio waves or cell phone signals, could cause the device to misfire and detect a false strike.

Another important function of the Lightning Strike Detector is the ability to record the time of the lightning strike. Time in this definition is the exact date on which the strike occurred along with the exact time of day. In order to do this a precise clock and calendar would need to be used. The clock would have to run all the time to ensure that the time was accurate. Another issue with this function would be recording the time a strike occurred on a microcontroller and then displaying it on a screen. This would require a microcontroller to be able to communicate with both the clock and the screen at the same time.

The next function would be the ability to count several different lightning strikes. There is a possibility that the building this device is on can be struck more than once in a month or year. This requires the Lightning Strike Detector to be able to count and record many strikes. In order to record many strikes a microcontroller must be programmed to hold an array of memory. This array will allow the user to be able to locate all of the recorded memory. To make the device more user friendly there will be a scroll feature that will allow the user to see all of the recorded memory on a display screen.

Another important function of the device is the ability to withstand the weather elements. This includes temperature, precipitation and lightning. The materials chosen for the enclosure are very important to this subsystem. The material must be waterproof, but also should have the ability to shield the circuitry from the electrostatic discharge of lightning. A faraday cage could be used to protect the circuitry from lightning strikes. A faraday cage consists of a conductive material that completely surrounds the circuit and is also grounded.

Finally, the Lightning Strike Detector should be long-lasting. Long-lasting in this case means operating in the field for six months without having to change the battery. One way to help achieve this goal would be to use passive sensing elements. Passive elements would be ideal in the fact that they would not require additional power to operate them.
2.2 Selecting a Solution

Initially the design team had proposed four different design solutions. These design solutions included a magnetic field detector using an inductor, a magnetic field detector using a Hall Effect sensor, the implementation of reed switches and an electrostatic discharge detector. In order to select a solution from this list the design team constructed a selection matrix. This selection matrix is shown in table 1 and includes the design specifications, the importance of each specification, the different possible solutions and how these solutions compared to each design specification. To decide which design was ranked the highest the team multiplied the importance rating of each design specification with the ranking each design had for that design specification. Then each multiplied value was added up and totaled. The design with the highest value in the total row was then ranked the best.

After several weeks of testing the various proposed circuit designs, the design team decided to go with the magnetic field detection approach. This decision was made due to the fact that the magnetic field detector ranked the highest amongst all of the designs. From analyzing the selection matrix the design team realized that the main differences between the designs were sensing range and if the sensing element was passive or active.

The functional range of the magnetic field detector using an inductor approach is what makes it different from other designs. The design team believes that this approach will be successful due to the nature of magnetic fields. Magnetic field strength drastically decreases as a sensor is moved away from the source. Other lightning characteristics, such as electrostatic discharge, may remain high in magnitude even at large distances. This would cause the device to misfire and count strikes that did not hit the building. With this in mind it would be easier to tune the range of the device if it were measuring a magnetic field.

Faraday’s Law states that a change in the magnetic field surrounding a coil will induce a voltage across the coil. This law provides evidence to the team that if a lightning bolt struck a grounding rod, the current that would flow through the rod would produce a magnetic field and change the magnetic field surrounding a coil. This would produce a voltage that could be amplified and sent to a microcontroller to indicate that there was a lightning strike.
Table 1: Selection Matrix

<table>
<thead>
<tr>
<th>Engineering Criteria</th>
<th>Importance</th>
<th>Magnetic Field Detector Using Inductors</th>
<th>Magnetic Field Detector Using Hall Effect Sensors</th>
<th>Reed Switches</th>
<th>Electrostatic Discharge Detector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small/Lightweight Design</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>Withstands Elements</td>
<td>5</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Display Data on LCD Screen</td>
<td>5</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>LEDs to Indicate Functionality and Stored Data</td>
<td>2</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>User Settable and Re-Settable</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>User Friendly</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Operational up to 6 Months in the Field</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>Device Parts &lt; $25</td>
<td>3</td>
<td>9</td>
<td>9</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>Operates at Close Range Only (~12 in.)</td>
<td>5</td>
<td>9</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Passive Sensing Element</td>
<td>5</td>
<td>9</td>
<td>1</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td>273</td>
<td>191</td>
<td>239</td>
<td>221</td>
</tr>
</tbody>
</table>

2.3 Budget

One objective of this project was to make a low cost device that can be mass-produced and sold for a profit. Thus, the cost of all parts of the Lightning Strike Detector kept in mind and optimized throughout the design process. The team had a total of $500 in budget for this project. The cost of the Lightning Strike Detector unit is designed to be $89.46. The Enclosure, Arduino UNO and LCD will take up most of the budget.

The design team made an initial estimate for the project of about $75. A table breaking down the initial cost estimate made by the design team is shown in table 2. This price accounted for an economy of scale. The corporate sponsor, Instrumented Sensor Technology, asked for the design team to build a device that would cost under $25 to make. The design team realized that a prototype would require a higher cost due to the small quantity that components were being purchased in. The cost for an LCD screen and enclosure alone would surpass the $25 requirement. More work would need to be done to optimize the cost of the device to bring the production cost down.
# Table 2: Initial Cost Estimate

<table>
<thead>
<tr>
<th>Item</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensing Elements</td>
<td>$ 5.00</td>
</tr>
<tr>
<td>Microcontroller board</td>
<td>$ 25.00</td>
</tr>
<tr>
<td>Enclosure</td>
<td>$ 20.00</td>
</tr>
<tr>
<td>LCD Screen</td>
<td>$ 18.00</td>
</tr>
<tr>
<td>IC's</td>
<td>$ 5.00</td>
</tr>
<tr>
<td>Resistors/Capacitors</td>
<td>$ 2.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>$ 75.00</strong></td>
</tr>
</tbody>
</table>

## 2.4 Gantt Chart

![Gantt Chart](image_url)

Figure 2: Gantt Chart
Chapter 3: Technical Description of Work Performed

3.1 Hardware

3.1.1 Sensing and Amplifier Circuit

When lightning hits a grounding rod it will send current through it to ground. This current will produce a magnetic field around the rod strong enough to be detected by an inductor. An inductor in a changing magnetic field will have an induced voltage across it according to equation 1.

\[
EMF = -NA\frac{\Delta B}{\Delta t} \quad \text{(Equation 1)}
\]

In this equation EMF represents the induced voltage, \( N \) is the number of turns in the inductor, \( A \) is the cross-sectional area of the inductor, \( \Delta B \) is the change in the magnetic field due to the lightning strike and \( \Delta t \) is the duration of the lightning strike. A typical lightning strike can produce a magnetic field of 32.8 Gauss to 656 Gauss at a foot away from the grounding rod. This magnetic field strength along with the 10 mH inductor the design team selected would induce a voltage in the low millivolt range. Due to this the design team added an amplifier.

The team decided to use an instrumentation amplifier for the design. This was chosen because instrumentation amplifiers have good Common Mode Rejection Ratios which is ideal for the application.
for low voltage applications. Instrumentation amplifiers also allow for a large gain. The amplifier that was chosen was the INA128P. This particular amplifier had a gain that was decided by one resistor value. This made prototyping and fine tuning much easier. The gain equation for the INA128P is shown in equation 2.

\[
\text{Gain} = 1 + \frac{50K}{RG}
\]  

(Equation 2)

In this equation \( RG \) is the resistor that changes the gain. Due to the fact that the induced voltage across the inductor is very small, a large gain was required. The team decided to use a 12 Ohm resistor for \( RG \) which would create a gain of 4168. This gain would be large enough to amplify the small induced voltages.

Figure 3 shows a schematic of the final amplifier circuit. It consists of an inductor, a resistor, the INA128P and two power sources. The power sources will be two 9 V batteries. One will have to be connected in reverse to account for the negative supply voltage. The output of this circuit will become the input to the one-shot multivibrator circuit.

3.1.2 One-Shot Multivibrator

![One Shot Multivibrator](image)
Due to the short duration of a lightning strike the microcontroller might be too slow to capture a strike. Due to this the design team implemented a one-shot multivibrator. A one-shot multivibrator is a circuit that will receive an input pulse and then output another pulse, but with a longer pulse width. Also, a lightning strike varies in magnitude, which means the induced voltage across the inductor will vary as well. To ensure that the input voltage to the Arduino UNO is not too high the one-shot multivibrator will limit the voltage.

The schematic for the one-shot multivibrator is shown in figure 4. This circuit is made of two comparators and a RC charging portion. The comparator chosen for this application was the LM339. The LM339 has four comparators on one chip, which made it desirable for this application. Each comparator has a reference voltage tied to its positive terminal. The first comparator has a reference voltage of 682 mV and the second comparator has a reference voltage of 7.16 V. When there is no input to the negative terminal of the first comparator the difference between the positive and negative terminal will be positive. This will cause the output of that comparator will be pulled high by a pull up resistor. This will charge the capacitor at a rate according to equation 3.

\[ V_C = V_S (1 - e^{-t/RC}) \]  
(Equation 3)

In this case \( V_C \) is the voltage across the capacitor, \( V_S \) is the 9 V source, \( R \) is the resistor value, \( C \) is the capacitor value and \( t \) is time. When the capacitor is fully charged its voltage will be larger than the voltage of the positive of the second comparator. This causes the difference between the positive and negative terminal of the second comparator to be negative. A negative voltage difference will pull the low. This is the initial state of the one-shot multivibrator.

When there is an input into the one-shot multivibrator that is larger than 500 mV the output will be pulled low. This causes the capacitor to be shorted to ground. With the capacitor shorted to ground it will begin to discharge at a rate according to equation 4.

\[ V_C = V_S e^{-t/RC} \]  
(Equation 4)
In this case there is no resistor in the RC circuit because the capacitor is short circuited to ground. A short circuit can to ground can be simulated by a very small resistor. A very small resistor will cause the capacitor to discharge very rapidly. This causes the voltage at the negative terminal of the second comparator to go lower than the second reference voltage. If the negative terminal is lower than the positive terminal there will be a positive voltage difference. A positive difference will cause the output to be pulled high. This output will continue to stay high until the negative terminal of the second comparator is higher than the positive terminal. This won’t happen until the capacitor recharges back up past 7.16 V. With the R and C values selected the voltage across the capacitor will not reach 7.16 V for 317 milliseconds. A larger or shorter pulse width can be made by changing the R and C values.

The one-shot multivibrator also provides protection for the Arduino UNO. The one-shot multivibrator can take a 36 V input before there are issues. The Arduino UNO, however, can safely receive a 5 V input. This requires the output of the one-shot multivibrator to have to be 5 V or less. To ensure this is the case a voltage divider was placed at the output of the second comparator. When the output of the second comparator goes high, it will only be pulled up to the value of the voltage divider. This provides a safe protection for the Arduino UNO while also provides a longer pulse width.

3.1.3 Arduino-LCD Interface (Hardware)

In order to record and display a lightning strike, the output of the multivibrator circuit was connected to an Arduino Uno microcontroller that writes the data to an LCD display. The Arduino Uno board consists of ATmega328 microcontroller, 14 digital input/output pins, 6 analog inputs, a 16 MHz resonator, a USB connection, a power jack, and an ICSP header. For this project the most important features of the board are the microcontroller, which allows for the building of programs in the JAVA programming language, and the digital input/output pins, which allows for the transfer of information between the board and the LCD display as well as the board and the multivibrator circuit.
The Arduino Uno microcontroller can be powered through a DC power jack, its USB connector, or through the \( V_{\text{IN}} \) pin on the board. The team selected to supply power to the board through the \( V_{\text{IN}} \) pin with a 9V battery, as it is rated at voltages between 7-12V. The Arduino Uno can also output a voltage of 5V or 3.3V when being powered, which is utilized to supply the LCD screen and various other parts of the circuit with 5V.

A 20x4 LCD screen was selected as the display, which consisted of 16 pins that control various features of the screen. The LCD screen can operate in both 8-bit mode and 4-bit. For this project the LCD display was wired to operate in 4-bit mode, as it requires fewer pins and is generally easier to use. A table that shows the pins and describes each function can be seen in Table 3 below. In order to power the display the 5V power output of the Arduino Uno was wired to the \( V_{\text{DD}} \) pin, while the \( V_{\text{SS}} \) pin was wired to one of the Arduino Uno’s ground ports.

The contrast of the screen can be adjusted by wiring a variable resistor to \( V_{0} \) located at pin 3 on the screen. While this method was employed using a 10k variable resistor during all testing, it is replaced with a 1.8k resistor wired to ground in the final design. This is done in order to avoid the variable resistor accidentally getting turned, causing an undesirable change to the contrast of the screen. The RS (Select display data or instructions) and E (Read/Write Enable) pins are wired to the digital pins 12 and 11 respectively on the Arduino Uno board.

Figure 5: An Arduino Uno Board
R/W (Read/Write Select) pin was wired directly to ground, ensuring that the screen is constantly in write mode in order to be able to consistently update.

Table 3: Pinout of LCD Display

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Symbol</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Vss</td>
<td>Ground</td>
</tr>
<tr>
<td>2</td>
<td>VDD</td>
<td>Power supply</td>
</tr>
<tr>
<td>3</td>
<td>V0</td>
<td>Power Supply for LCD</td>
</tr>
<tr>
<td>4</td>
<td>RS</td>
<td>Select Display Data(“H”) or Instructions(“L”)</td>
</tr>
<tr>
<td>5</td>
<td>R/W</td>
<td>Read or Write Select Signal</td>
</tr>
<tr>
<td>6</td>
<td>E</td>
<td>Read/Write Enable Signal</td>
</tr>
<tr>
<td>7</td>
<td>DB0</td>
<td>Display Data Signal</td>
</tr>
<tr>
<td>8</td>
<td>DB1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>DB2</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>DB3</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>DB4</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>DB5</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>DB6</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>DB7</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>LED – (K)</td>
<td>Please also refer to 6.1 PCB drawing and description.</td>
</tr>
<tr>
<td>16</td>
<td>LED + (A)</td>
<td>Please also refer to 6.1 PCB drawing and description.</td>
</tr>
</tbody>
</table>

To interface with the LCD in 4-bit mode the Arduino only needs to be connected to pins DB4-DB7, which will connected to digital pins 5-2 respectively. Pins 15 and 16 on the LCD screen are used to power a backlight in the screen. These pins were chosen to be left unwired in the final design of the project by the team. This is because the screen is easy to read without a backlight in most environments and the power consumption of the screen is significantly reduced when this feature is not used.
In order to register a strike the analog input pin A0 has been wired to the output of the multivibrator circuit. As discussed earlier in this report, the multivibrator circuit outputs 5V for approximately one second in the event of a lightning strike. The Arduino Uno microcontroller has been programmed to print the lightning strike count value, date, and time of the strike to the screen when this voltage change is measured across pin A0. Specifics on how this is coded will be discussed later in this project under the software section. In order to keep track of the current date and time a DS1307 Real Time Clock (RTC) Module was wired to the Arduino Uno using analog pins A4 and A5. These pins were wired to the SDA and SCL pins of the RTC respectively. The RTC is powered by the 5V output of the Arduino Uno when the board is on, and has a 3V backup battery that keeps the time when not being powered by the Arduino Uno. The RTC can be set automatically when the Arduino Uno is connected to a computer through its USB connector, or set manually through code provided by an Arduino library. The RTC was chosen to be set manually, as setting the clock automatically proved to be much less accurate than doing it manually. The code used to manually set the RTC can be found at the appendix at...
the end of this report. Once set the RTC is advertised to be able to keep accurate time up to a minimum of 9 years.

![Figure 7: DS1307 RTC Module](image)

In order to manipulate the data appearing on the screen, the remaining three analog pins have been wired to push buttons mounted on the surface of the device. Pins A1 and A2 have been connected to buttons used to scroll the screen data up or down. Since the screen is only a 20x4 display, only 3 rows of data and the header row can be displayed at once despite the Arduino Uno having enough memory to store 83 strikes. These buttons allow the user to scroll up or down to be able to view all of the data that has been recorded and displayed on the screen. Pin A3 is connected to a reset button, which erases all of the data stored on the screen, with the exception of the header row at the top. The buttons are wired to have one end attached to the analog pin and the other end attached to a 5V bus powered by the output of the Arduino Uno board. When the button is pushed, 5V are applied to the pin and the Arduino Uno performs the desired action. Once again, specifics on how these features are programmed will be touched upon later in the report. All of these analog pins, as well as pin A0, are attached to 1k pull down resistors to ensure their resting voltage is 0V. This is because when the analog pins are left floating applying 5V to one of them can sometimes leak to other nearby pins, triggering them high when they are not supposed to be.

The final two elements wired to the Arduino Uno are a red and green LED, which are attached to the digital pins 7 and 8. The green LED has been selected to turn on when power is applied to the Arduino Uno, indicating to the user that the device is turned on. The red LED is programmed to blink when data is written to the screen, which indicates to the user that there is data recorded on the display that the user has not seen. Pressing the scroll up, scroll down, or
reset button turns this LED off, because pressing any of these buttons indicates that the user has seen the display and is up to date on the data that has been recorded. In order to limit the power consumption of the LEDs, a 560 Ω resistor and 220 Ω resistor have been wired between the Arduino Uno board and the green and red resistor respectively.

![Figure 8: Final Prototype Setup](image)

### 3.1.4 Enclosure

A faraday cage is used to protect our circuit from any interference that could trigger the sensing circuit of the device. This enclosure is necessary as the EMP radiated from a lightning strike has a large electrical component to it, which could cause the device to malfunction. This type of interference is can also be produced by any electronics functioning in the area, which could result in the device being triggered when no lightning strike has occurred. To confirm the shielding potential of faraday cage the team performed tests utilizing the Van de Graaff generator provided, and are described in greater detail in the “Testing” section of this report.
The faraday cage the team decided to utilize is a 7.4x4.7x2.2 aluminum box. Because aluminum is a conductor it will prevent any electrical field from reaching the circuit inside while still allowing a magnetic field to pass through due to its non-magnetic properties. The ECE shop in the Engineering Building provided aid in drilling out holes in the enclosure for the LCD display, user push buttons, and LED indicators. Figure 9 shows the final enclosure.

![Figure 9: Final Enclosure](image)

3.2 Software

3.2.1 Arduino-LCD Interface (Software)

In order to program the Arduino Uno microcontroller to carry out the various functions described above Arduino Software (IDE) version 1.6.3 was utilized. In order to use this environment to program the Arduino Uno microcontroller the JAVA programming language was used. In order to create an executable cyclic program that can be run by the Arduino, two functions need to first be defined: A setup (Setup()) and Loop()). The setup function is run once at the beginning of the program and is used to initialize settings. The loop function is called
repeatedly while the board is powered and where a majority of the work was done for this program.

```c
void setup() {
    // Set up the LCD's number of columns and rows:
    lcd.begin(20, 4);

    // The function to get the time from the RTC
    Serial.begin(9600);
    setSyncProvider(RTC.get); // the function to get the time from the RTC
    if (timeStatus() != timeSet)
        Serial.println("Unable to sync with the RTC");
    else
        Serial.println("RTC has set the system time");

    // Clear display
    lcd.clear();

    // Print a message to the LCD.
    lcd.print("Cnt");
    // set the cursor to first row, 9th column
    lcd.setCursor(4, 0);
    lcd.print("Date");

    lcd.setCursor(13, 0);
    lcd.print("Time");

    // Initialize digital pins as outputs
    pinMode(led1, OUTPUT);
    pinMode(led2, OUTPUT);
}
```

Figure 10: Setup Function of Arduino Uno Program

Figure 10 above shows the setup function used in the Arduino Uno program of the project. It begins by defining the display size of the LCD to the Arduino as 20x4. The next section of code is dedicated to retrieving the time set to the RTC clock to ensure that the Arduino Uno prints the current time and date to the screen. The program then prints the header row to the LCD display, which consists of “Cnt”, “Date”, and “Time” columns. These will display the strike count, current date, and current time respectively. Lastly, the setup loop sets digital pins 7 and
8, which are labeled as led1 and led2, as digital outputs instead of inputs. This will allow the pins to drive the voltage across the LEDs wired to the pins high or low, which will turn the LEDs on or off.

```c
void loop() {

  //Turns on LED 1, indicating device is on
  digitalWrite(led1, HIGH);

  //Reads analog value at pin A0
  sensorValue = analogRead(A0);

  //Determines if analog value is high or low
  if (sensorValue > 500) {
    // set the cursor to column 0, line 1
    // (note: line 1 is the second row, since counting begins with 0):

    //Update arrays with current data time
    mn[c_val] = month();
    dy[c_val] = day();
    yr[c_val] = year() % 100;
    hr[c_val] = hour();
    CheckAMPm(c_val); //adjust 24 hour to AM or PM
    mi[c_val] = minute();

    c_val = c_val + 1; //Increase count value
    if (c_val > 3)
      toprow = c_val - 2; //auto scroll
      UpdateLCD();

    res = 1;
    sensorValue = 0;

delay(1000);
  }
}
```

Figure 11: Beginning of Loop Function of Arduino Uno Program

Figure 11 above shows the first part of the loop function used in the Arduino Uno program of the project. The first line of code sets the output of the digital pin of the Arduino Uno high, which turns on the green LED. This indicates to the user that the device is currently running. Next it sets a variable, sensorValue, to the value that is measured at the analog input
pin A0. The maximum recommended value that can be applied to the analog input pins of the Arduino Uno board is 5V. This voltage is converted to a digital value between 0 and 1024 bits, with 0V corresponding to 0 bits and 5V corresponding to 1024 bits. As can be seen in the screenshot above, the analog pin A0 senses for a signal greater than 500 bits, which corresponds to approximately 2.5V (2.44V exactly). Because the multivibrator circuit outputs 5V in the event of a strike, this value is low enough to record any instance of a strike while remaining high enough to filter out any noise that could potentially cause a false strike to occur.

If a value of more than 2.5V is measured at the input pin A0 the program enters the “if” statement. The first thing that is done inside the “if” statement is to update the arrays that have been created to store the current month, day, year, hour, and minute. A variable, “c_val” has been created to record the strike count. This variable is initialized at 0 and increments by one every time a strike is registered. The time arrays are updated with respect to a certain value of “c_val”, thus ensuring each strike is associated with a certain time and date. A function called “CheckAMPM” is also called in order to convert the default 24 hour time format of the RTC to the more common 12 hour format. This function can be seen in Figure 12 below.

```c
void CheckAMPM(int index) {
    int hour12;
    bool PM = false;
    hour12 = hr[index];
    PM = hour12 >= 12;
    if (PM && (hour12 != 12))
        hour12 = hour12 - 12;
    if (hour12 == 0)
        hour12 = 12;
    hr[index] = hour12;  //update hour array
    pm[index] = PM;
}
```

Figure 12: The “CheckAMPM” Function

Following this, the program increments “c_val” and sets the value of “toprow”. The variable “toprow” was created to define the top row of the data array. This is necessary because the array is large enough for 83 rows, while the screen can only display three at a time. This allows for the screen to scroll down as strikes occur, which ensures that the screen is
defaulted to display the most recent three strikes. For example, if there are 10 strikes recorded
the LCD display will show rows 8, 9 and 10. The program then calls the “UpdateLCD” function,
which is the function responsible for actually writing text to the screen. The variable res
was created to turn on the flashing red LED and will be discussed further in this report. Finally,
before the “if” statement is closed a one second delay is added, ensuring that one strike is not
registered as multiple by the program.

```c
void UpdateLCD(void) {
    int i;

    for (i=0; i<3; i++) {
        lcd.setCursor(0, i+1); // row are 0-3, header is row zero
        stringOne = ""; // start new strike line
        if (toprow+i > c_val)
            break;
        BuildString(" ",0,3,toprow+i); // add c_val to stringOne;
        BuildString(" ",1,2,mn[toprow-l+i]); // add month to stringOne;
        BuildString(" ",1,2,dy[toprow-l+i]); // add day of month to stringOne;
        BuildString(" ",1,2,yr[toprow-l+i]); // add year to stringOne;
        BuildString(" ",1,2,hr[toprow-l+i]); // add hour to stringOne;
        BuildString(" ",1,2,mi[toprow-l+i]); // add minute to stringOne;
        if ( pm[toprow-l+i] )
            stringOne = stringOne + "PM``
        else
            stringOne = stringOne + "AM``;
        lcd.print(stringOne);
    }
}
```

Figure 13: The “UpdateLCD” Function

```c
void BuildString(String sym, int symcnt, int valcnt, int val) {
    // utility function: append to str # of spaces followed by val with min length of digits
    String tmpstr = "";
    int i = 0;

    while (i++ < symcnt)
        stringOne = stringOne + sym; // add symcnt symbols to end of string

    tmpstr = String(val);
    while (tmpstr.length() < valcnt)
        tmpstr = "0" + tmpstr; // add val to string with correct number of digits
    stringOne = stringOne + tmpstr;
}
```

Figure 14: The “BuildString” Function
Above in Figures 13 and 14 are the “UpdateLCD” and “BuildString” functions, which are used to print text to the LCD display. The “UpdateLCD” function begins with a “for” loop sets the range of the data array that will be displayed on the screen (Such as rows 1-3, 2-4, 3-5, etc.). The “BuildString” function is then called multiple times, each time with a new value to add to the row that is being constructed. A row is built by adding each desired value one to an empty string, called “stringOne”, one at a time. This begins by adding the strike count (“c_val”, represented by “toprow+i” in the “UpdateLCD” function to ensure the top row of the array is correct), followed by the current date (“mn[]”, “dy[]”, and “yr[]”), and finishing with the current time (“hr[]”, “mn[]”). The “BuildString” function also pads each variable added with zeros to make it the length of each value constant string length regardless of count, date, or time (For example, the date 4/5/15 would be printed as 04/05/15). Once “stringOne” has been constructed the entire row is printed to the screen.

```c
if (res > 0) {
  if (oncnt == 0 && offcnt == 0) {
    // Turns on LED 2, indicating data has been recorded
    digitalWrite(led2, HIGH);
    // Wait for a second
    oncnt = 750;
    offcnt = 500;
  }
  else if (oncnt == 0 && offcnt != 0) {
    digitalWrite(led2, LOW);
  }
}
else if (res < 1)
  digitalWrite(led2, LOW);

if (oncnt > 0)
  oncnt--;
if (offcnt > 0 && oncnt == 0)
  offcnt--;
```

Figure 15: LED Code

Figure 15 above shows the code that was used to control the flashing red LED described earlier in the report. The code begins by using an “if” statement to check the value of the “res” variable. Recall that this variable is set to equal 1 in the bit of code that detects a strike. Once a
strike is detected, this “if” statement is entered and another if statement checks the values of “oncnt” and “offcnt”. These are variables that are set to certain values inside this “if” statement and decrement with every cycle of the loop. This is used to control the rate at which the LED will flash without using a delay, since a delay will cause the loop function to temporarily stop working with every cycle of the loop. If both of these values are 0, the digital pin connected to the LED is driven high and the LED is turned on. These variables are then set to 750 and 500 respectively and begin to count down back to 0. Once “oncnt” reaches 0 the LED is turned off until “offcnt” reaches 0, at which point the LED turns back on. Another “if” statement is set to turn off the LED until a new strike is registered if “res” is set to 0. This will occur if one of the three push buttons (scroll up, scroll down, or reset) is pressed.

```c
sensorUP = analogRead(A1);
sensorDN = analogRead(A2);

if ((sensorUP > 500) && (uppressed == false)) {  
    sensorValue = 0;
    uppressed = true;
    res = 0;
    ScrollUP();
}  
else if (sensorUP < 300) {  
    sensorValue = 0;
    uppressed = false;
}

if ((sensorDN > 500) && (downpressed == false)) {  
    sensorValue = 0;
    downpressed = true;
    res = 0;
    ScrollDN();
} 
else if (sensorDN < 300) {  
    sensorValue = 0;
    downpressed = false;
}

if (analogRead(A1) > 900) {  
    lcd.setCursor(0, 1);
    lcd.println(BlankLine);
    lcd.setCursor(0, 2);
    lcd.println(BlankLine);
    lcd.setCursor(0, 3);
    lcd.println(BlankLine);
    c_val = 0;
    toprow = 1;
    res = 0;
}  

delay(1);
```

Figure 16: End of Loop Function
As mentioned earlier in the report, the device has three buttons used for user interface with the device. These buttons allow the user to scroll through or clear the data displayed on the screen. The scroll functions work by measuring the voltage at the analog pins A1 and A2. If the button is pushed and 5V are applied to the pin the “if” statement is entered, calling either the “ScrollUP” or “ScrollDN” function. These functions first check to see if it is possible to scroll up or down by checking the value of “toprow” against the value of “c_val”. If it is possible to scroll up or down (i.e. if the bottom top row of the array isn’t the first strike recorded or the bottom row isn’t the most recent strike recorded) the functions either increment or decrement the value of “toprow” and call the “UpdateLCD” function. The reset button works in similar fashion to the scroll buttons. However, in this case if the “if” statement is entered the current data lines being displayed by the LCD are overwritten with blank lines and the variables “c_val”, “toprow”, and “res” are reset to their initial values. To reference the full code used in this project, refer the appendix at the end of this report.

Chapter 4: Test Data with Proof of Functional Design

4.1 Test Procedures

Proper testing of the lightning strike was done completely through simulation, due to the dangerous nature of close-impact lightning strikes. As a substitute, a variety of equipment was used to simulate the occurrence of a nearby lightning strike. During all testing phases it was
the primary goal of the team to replicate the EMP given off by a lightning strike as closely as possible in order to ensure the most accurate results.

A Van de Graaff generator provided by the team’s corporate sponsor was used for testing early in the design phase. This device was selected for testing as it allowed the team to replicate a lightning strike, and thus the EMP, on a much smaller and safer scale. However, this method proved to be ineffective at testing the team’s design as it was not able to produce a large enough magnetic field. Equation 5 below was used by the team to calculate the approximate value of the magnetic field radiating from a grounding rod assuming that an average strike carries 30 kA of current, where “B” is the magnetic field strength (T), “μ” is the permeability of free space (4πx 10^-7 T * m/A), “I” is the current passing through the grounding rod (A), and “r” is the distance away from the grounding rod (m).

\[ B = \frac{\mu I}{2\pi r} \]  

(Equation 5)

Using this equation it was determined that the magnetic field strength at a distance of one foot away from the grounding rod would be approximately 19.7 mT, or 196.85 gauss. Because the Van de Graaff generator could only produce micro amps the magnetic field strength that it was able to produce was nowhere close to what would actually occur in the event of a lightning strike.

The Van de Graaff generator was, however, useful in testing the shielding aspect of the design. While not able to generate a large enough magnetic field to be detected by the sensing element of the circuit, the electric field radiated from each strike was strong enough to be picked up by the leads and wires of the circuit. This would often cause the device to trigger even when disconnected from the sensing circuit. The proposed solution to filter out this part of the EMP was to place the circuit in a Faraday cage, as described earlier in the report. The team was able to create a very basic faraday cage out of a cardboard box wrapped in aluminum foil in order to test its effect. Using this, it was found that a faraday cage would be successful in protecting the device from any outside interference not due to the magnetic field, as the number of false triggers was greatly reduced when the circuit was placed in the box.
While considered, a Tesla Coil was decided against for testing purposes. While a Tesla Coil has a much greater potential for magnetic field strength than the Van de Graaff generator, the fields are still insufficient of being considered comparable to a lightning strike. Proper simulation would have required a current threshold of several thousand amperes. The Tesla Coil available to the design team was limited to 20 amperes, which is still considerable too low to generate the desired magnetic field.

The majority of testing was done through the use of a 12600 Gauss bar magnet. This method of testing allowed for the team to test the device with a magnetic field comparable to what is radiated from a grounding rod when struck by lightning. The testing method the team employed with the bar magnet consisted of waving it at various distances away from the sensing inductor. As the magnetic field produced by a magnet dissipates by a factor $r^3$, where $r$ is its distance away from a desired point, it was possible for the team to determine the magnetic field strength at the inductor due to the bar magnet at each distance measured. These calculations were done using an online calculator, which took into account both the size and shape of the magnet to get the most precise values. These values can be found in Table 4 in the next section, as well as the corresponding strength of the magnetic field radiating from a grounding rod when as is struck by lightning of various magnitudes. Using this information the group could use the bar magnet to generate a magnetic field equivalent to what would be generated by a lightning strike.

The team took a few assumptions when employing this testing method. First of all, it was assumed that the device would be placed a foot away from the grounding rod. This distance was taken into account when calculating the expected magnetic field strength at the inductor that would result from lightning striking a grounding rod. The group also assumed that the faraday cage that would be placed around the final design of the device would be successful in filtering out most interference that could cause the device to function improperly. This assumption was made due to the success of the team constructed faraday cage had at performing this function in earlier trials.
4.2 Project Functionality

The final product developed by the design team was determined to be a success on many aspects. The main objective of the design team was to create a lightning strike detector that would only detect lightning from a short distance away. In this project a short distance refers to within a foot. As the device moves further away from the source of the magnetic field, the strength of the magnetic field decreases. This required the design team to perform several tests at different distances. Table 4 shows the magnetic field due to a 12,600 Gauss magnet, the current required to produce the same magnetic strength at a foot away and success of the design versus distance. This table shows that the proposed design will detect a lightning strike that produces a current of 9.01 kA at a foot away. The design team determined this to be very successful due to the fact that lightning, on average, produces current from 5-100 kA.

<table>
<thead>
<tr>
<th>Distance (Inches)</th>
<th>Magnetic Field Due to 12600 Gauss Magnet (Gauss)</th>
<th>Current Needed from Lightning at 1 ft. (kA)</th>
<th>Success?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1404.6</td>
<td>214.1</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>584.3</td>
<td>89.0</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>284.3</td>
<td>43.3</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>155.3</td>
<td>23.7</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>92.6</td>
<td>14.1</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>59.2</td>
<td>9.02</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>39.9</td>
<td>6.08</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>28.0</td>
<td>4.27</td>
<td>No</td>
</tr>
</tbody>
</table>

The corporate sponsor of the Lightning Strike Detector team required the device to detect lightning within a foot of the source. The final proposed design will be able to meet those expectations knowing that the device works at a maximum distance of a foot away. If the device is moved any closer to the source of the strike it will still work to expectations. The design team also meet other design specifications such as the ability to display data on an LCD screen, the use of passive sensing elements, a small and lightweight design and the use of LED’s to indicate power ON and if data is stored. Overall the corporate sponsor, Instrumented Sensor Technology, was pleased with the design and the attention to the design specifications.
4.3 Sustainability

All components of the design are sustainable indefinitely and should not have any varying functionality over time. However, the device was design to be powered by three batteries, which can cause the functionality of the device to decline as they discharge to a certain point. The primary issue relating to device sustainability the team experienced occurred with the battery responsible for powering the multivibrator circuit.

As you may recall from earlier in the report, the multivibrator circuit functions by comparing the output of the sensing circuit to a reference voltage set by the team. The output of the sensing circuit rests at a lower voltage than the multivibrator reference voltage, but when the sensing circuit detects a strike its output becomes greater than the reference voltage and the multivibrator goes high. However, as the battery that powers the multivibrator discharges, the reference voltage decreases with it. Once the battery discharges to a certain point the resting voltage of the sensing circuit output sits at a higher voltage than the multivibrator reference voltage. This results in the multivibrator remaining in its “on” state and the device falsely begins to detect lightning strikes. As a preventative measure for this issue the user should ensure that the batteries powering this section of the device are sufficiently charged before use.

Chapter 5: Final Cost, Schedule, Summary and Conclusion

5.1 Summary

The lightning strike detector project required sufficient knowledge in many electrical engineering categories. To begin the design team did extensive research on the characteristics of lightning strikes. This research provided the team with many ideas on how to safely sense a lightning strike from a foot away. Throughout the design process the team tested these different ideas. Through these tests the design team found that sensing an induced magnetic field due to current flowing through a grounding rod would be the best way to achieve the project goal.

At first the design team had difficulty generating a magnetic field strong enough to simulate a lightning strike. This led the team to research rare earth magnets and compared
their magnetic field strength to the strength of a lightning strike. Their strength ended up being comparable. This breakthrough allowed the team to perform safe tests with the magnet instead of having to generate large amounts of current with the lab equipment.

With an approach in mind the design team worked to build a circuit that would detect a magnetic field and indicate a microcontroller. The most difficult part of the circuit was finding a circuit element that would detect a magnetic field, but also require little power. Circuit components such as inductors, hall effect sensors and magneto resistors were tested only to find that inductors best fit the requirements.

Inductors have an induced voltage across them when there is a changing magnetic field in its presence. Unfortunately, this voltage is very small and will not be able to set off the microcontroller. To fix this issue the team added an amplifier circuit to boost the detected signal to an appropriate value of 1 V or higher. This amplifier was also chosen specifically to reduce the amount of external noise present in circuit.

The design team also wanted to ensure that the detected signal would be long enough to be seen by the microcontroller. For this issue the team implemented a one-shot multivibrator circuit whose purpose is to increase the pulse width of an incoming signal. The one-shot multivibrator also provides voltage protection for the microcontroller as it will only output 0 V or 5 V.

Finally, the team needed to display the detected lightning strike. To do this the design team used an LCD screen that would display the amount of strikes that were detected and at exactly what date and time. The final design successfully detects a magnetic field of amplitude similar to a lightning strike at about a foot away. It then amplifies the signal and increases its pulse width. Then the signal is passed to the microcontroller where it can be read and sent to an LCD screen to display.

### 5.2 Final Costs

The final cost of the Lightning Strike Detector for a large scale production is $89.46. The breakdown of the cost is shown in figure 18. This cost breakdown shows where the money was spent, but also how much each component costs compared to the rest of the design. This cost
is misleading because there are many components that can be changed slightly to drastically reduce costs. For example, the Arduino UNO could be replaced by the ATMega328 and reduce the cost by $23.92. Also, the enclosure and LCD screen could be reduced in size, which will reduce the costs as well. In the end the design team made efforts to keep costs low and also researched different options to reduce costs.

Figure 19 shows the total prototyping cost that was accumulated throughout the design process. This cost accounts for various different approaches that were tried. Along with the components shown in figure 19 the design team utilized the Electrical Engineering Department parts shop and free samples from Texas Instruments to prototype and test the Lightning Strike Detector.

![Final Product Cost](image)

**Figure 18: Final Product Cost**
5.3 Future Improvements

While the team was successful in meeting most of the specifications outlined by the customer, the time constraint put on the project made it difficult to satisfy every requirement. In addition to this, the team feels that the current design could be improved given more time. First and foremost, the current casing for the device is not waterproof. Because the device will be outside and subject to rain while being used, this could be a major issue. Future designs will need some kind of waterproof covering in order to protect the device from bad weather.

Another issue with the current device is battery life. While the project requirements called for the device to be functional for up to six months in the field, the current design only has a battery life of a little over 100 hours. In order to improve the lifespan, the team has considered a few different options. First of all, different batteries could be selected to power the device. In the current design, the device is powered by three standard Energizer 9V batteries. These could all be replaced by batteries with a higher service hour rating, which would allow for the device to last longer in the field. Another option would be to use solar powered batteries in the device. Since the device will be outdoors when being used, this option could allow for the device to function indefinitely under the right conditions.

One aspect of the current design that could be improved upon in the future is the device sensitivity. While the team feels that the device sensitivity is adequate to meet the needs...
outlined by the customer, it is possible to make the sensitivity of the device tunable by the user. By adding a variable resistor or variable inductor the user could change the gain of op-amp used to amplify the signal incoming signal or change the range of the sensing element of the device. This can allow the user to make the device more or less sensitive to better suit their needs.

Finally, it was considered by the team numerous times to try and make the device transmit its data wirelessly to nearby computers instead of displaying information on an LCD screen. This would allow the user to download information to their phone or computer rather than going to the location of the device to read the information off of the LCD display. A mobile device application could even potentially be developed for the purpose of interfacing with the device. This could also have the potential to significantly increase battery life, as the LCD display circuit draws the most power from the batteries.

Appendix 1: Technical Roles, Responsibilities and Work Accomplished

Justin Bauer - Team Manager

The technical role that Justin played as a part of the Lightning Strike Detector team was designing the analog circuits. These circuits included an instrumentation amplifier and a one-shot multivibrator. These two sub-circuits were responsible for sensing an induced magnetic field due to a lightning strike, amplifying the signal, and then increasing the pulse width of the signal so that it can be read by the microcontroller.

In designing the instrumentation amplifier Justin researched various amplifiers so that he could select the one most equipped for this application. This included reading several datasheets and application notes to ensure that the correct instrumentation amplifier was selected. Once the instrumentation amplifier was selected Justin began running tests to find the appropriate gain needed from this amplifier. Justin was able to select a gain value that
allowed a magnetic field of similar magnitude of a lightning strike to be amplified to a useable level.

Designing the one-shot multivibrator consisted of several steps and trials. Justin began his design by testing a one-shot multivibrator circuit found on a Texas Instrument datasheet. This one-shot multivibrator, however, did not work for the required application. Justin then had to design a new one-shot multivibrator from scratch using concepts he had learned while testing the Texas Instruments circuit. Designing the new circuit required a lot of testing and interchanging components. Because of this Justin used the PSPICE software to test the circuit. This allowed Justin to select appropriate component values before putting them on a breadboard. The final design of the one-shot multivibrator was able to take an input signal and increase its pulse width by 317 milliseconds.

Justin was also able to contribute with testing the microcontroller software however, most of Justin’s technical contribution came with designing and testing analog circuits. Designing and testing analog circuits required a good amount of knowledge in various analog components and also in reading datasheets.

Zongheng Pu - Webmaster

The technical role of Zongheng in the group is Information seeker and idea contributor. The idea proposed by him including Using of Arduino for LCD display, LCD displayed pattern and size, Faraday cage usage, detecting method of inductor, etc. Since the MSP430 in our ECE480 is not that user friendly for the project to work on, Zongheng has used Arduino in his own project over past two years. Thus Zongheng provide the Arduino UNO for the lightning strike detector group to use and discuss with programmer Matt on LCD programming. Zongheng also proposed
original idea with group on LCD selection and the feature displaying, such as light strength and character array. About the Faraday cage, Zongheng suggests using aluminum foil as the way to protect our circuit from electric field, and ran the test in the beginning with first prototype while using van de Graaff as the current generator. As to the detecting method, the original idea provided by Zongheng is to wrap our detecting inductor around the grounding rod. Therefore, once the lightning strike occurs, the surge of current go through the lightning rod will trigger enough magnetic field to generate small amount of electrical signal which can be easily captured by placing inductor in this way. But the safety of device is also a drawback in this method. After several discussions and testing, the group decided to modify it to be placed into a faraday cage but close to grounding rod for receiving magnetic field.

Zongheng also work as recorder in the group. In order to keep the website update regularly, Zongheng needs to keep track of team process on both circuit designing and code programming. Taking recorded data and pictures for every capstone in Lightning strike detector group and post it on team’s website. While discussing with team designer, Zongheng also provide suggestion and possible solution to problems the team struggle at.

DeAndre Dawson - Document Prepare

The technical role that Deandre played as a part of the Lightning Strike Detector team was the document prepare. With the assistance of Justin all document outlines and document portions were rationed amongst the group. He also contributed to the initial development ideas for the circuit which involved using the hall-effect sensor, and some circuit analysis for trouble shooting. Another major role he played was with the development of the outer casing of the design. This design includes one power switch on the side, a precisely cut rectangle shape to fit the LCD, in the upper portion of the box, and five buttons for functionality. During the
development of our project each member played an essential role in making sure our project was complete in a timely fashion.

Adam McHale - Presentation Preparation

Adam’s technical role in the design of the lightning strike detector primarily consisted of calculating and generating the proper strength of a magnetic field. In order to properly represent the external conditions of a lightning strike, a comparably strong magnetic field needed to be generated.

By Ampere’s Law, a magnetic field is induced surrounding the flow of current with intensity directly proportional to that of the current flowing. An average lightning strike is expected to carry five to twenty-thousand amperes of current, equating to a variation in one to four times the magnitude of the expected surrounding magnetic field. Comparatively, the intensity of the magnetic field varies inversely proportionally with the distance at which the field is measured. This made it convenient to vary the strength of the magnetic field based on the anticipated current.

Induction of the magnetic field was initially attempted through the generation of arcing currents, functioning similarly to the way in which charges are transferred during a lightning strike. A Van de Graaff generator was used for this purpose, as charges build up on the surface of the generator only to be discharged in the form of arcing currents. This short-duration current possess the same properties as a lightning strike, inducing a magnetic field.

The current from the operation of the Van de Graaff generator proved to be insufficient in inducing a magnetic field comparable to that of a lightning strike, due to the low magnitude of current produced in electrical arcs. A more realistic approximation was found in the way of
permanent magnets. Comparative to the arcing currents of a lightning strike, permanent magnets produce a time-invariant magnetic field. In order to properly simulate a lightning strike, a permanent magnet must be rotated around the lightning strike detector at a desired radius in order to produce a time-varying magnetic field.

**Matt Clary - Lab Coordinator**

The technical role that Matt played in the Lightning Strike Detection team was the design and programming of the Arduino-LCD interfacing circuit. Specifically, this task was composed of configuring the Arduino Uno microcontroller monitor the output of the multivibrator circuit for indications of the lightning strike, record this data, and display the information on the LCD screen.

In designing this circuit, Matt referenced various sources detailing ways to interface an Arduino Uno microcontroller with an LCD screen, on both the hardware and software side. This included analyzing the data sheets for both the Arduino Uno and various LCD displays to ensure all wiring was correct. This step also involved researching a variety of LCD displays to ensure the group selected one optimal for this project. After all relevant information was acquired, Matt constructed the Arduino-LCD circuit on a proto-board and began working on programming the Arduino Uno microcontroller.

Most of the time that Matt spent working on the project this semester went toward programming the Arduino Uno microcontroller to display the current time, date, and count number of the lightning strike on the LCD display. In order to complete this task a variety of tutorials and online forums were referenced in order to become familiarized with the Java programming language and learn techniques on how best to construct various functions. In
order to display the actual time and date of each recorded lightning strike a real time clock needed to be wired to the microcontroller. Matt was responsible for determining which Real Time Clock module should be used by the team and interfacing it with the Arduino Uno microcontroller. This also included finding code that designed to set the clock to the current date and time.

Once all prototyping and testing was completed, Matt began working on packaging the final design with the rest of the team. This involved the construction of the Arduino-LCD circuit on a solderable circuit board and mounting the circuit inside of the faraday cage.

Appendix 2: Literature and Website References

- High Voltage Engineering Fundamentals (Second Edition)  
  E. Kuffel, W.S. Zaengl and J. Kuffel
Appendix 3: Technical Attachments

Arduino Code

// include the library code:
#include <LiquidCrystal.h>
#include <Time.h>
#include <DS1307RTC.h>
#include <Wire.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

#define ASIZE 83

//Variable declaration
int c_val = 0;
int c_val_old = 0;
int res = 0;
int sensorUP = 0;
int sensorDN = 0;
int sensorValue = 0;
int led1 = 8;
int led2 = 7;
int oncnt = 0;
int offcnt = 0;
bool uppressed = false;
bool dnpressed = false;
String stringOne = "";
String blankline = "                      ";
int mn[ASIZE];
int dy[ASIZE];
```cpp
int yr[ASIZE];
int hr[ASIZE];
int mi[ASIZE];
bool pm[ASIZE];
int toprow = 1;

void setup() {

    //Set up the LCD's number of columns and rows:
    lcd.begin(20, 4);

    //The function to get the time from the RTC
    Serial.begin(9600);
    setSyncProvider(RTC.get);  // the function to get the time from the RTC
    if(timeStatus()!= timeSet)
        Serial.println("Unable to sync with the RTC");
    else
        Serial.println("RTC has set the system time");

    //Clear display
    lcd.clear();

    // Print a message to the LCD.
    lcd.print("Cnt");
    //set the cursor to first row, 9th column
    lcd.setCursor(4, 0);
    lcd.print("Date");
    lcd.setCursor(13, 0);
```
lcd.print("Time");

//Initialize digital pins as outputs
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);
}

void loop() {

//Turns on LED 1, indicating device is on
digitalWrite(led1, HIGH);

//Reads analog value at pin A0
sensorValue = analogRead(A0);

//Determines if analog value is high or low
if (sensorValue > 500) {
    // set the cursor to column 0, line 1
    // (note: line 1 is the second row, since counting begins with 0):
    //Update arrays with current data time
    mn[c_val] = month();
    dy[c_val] = day();
    yr[c_val] = year()%100;
    hr[c_val] = hour();
    CheckAMPM(c_val); //adjust 24 hour to AM or PM
    mi[c_val] = minute();

    c_val = c_val + 1; //Increase count value
    if ( c_val > 3)
toprow = c_val-2; //auto scroll
UpdateLCD();

res = 1;
sensorValue = 0;
delay(1000);
}

if (res > 0) {
  if (oncnt == 0 && offcnt == 0) {
    //Turns on LED 2, indicating data has been recorded
digitalWrite(led2, HIGH);
    //Wait for a second
    oncnt = 750;
    offcnt = 500;
  }
  else if (oncnt == 0 && offcnt != 0) {
    digitalWrite(led2, LOW);
  }
}
else if (res < 1)
  digitalWrite(led2, LOW);

if (oncnt > 0)
  oncnt--;
if (offcnt > 0 && oncnt == 0)
  offcnt--;
sensorUP = analogRead(A1);
sensorDN = analogRead(A2);

if ((sensorUP > 900) && (uppressed == false)) {
    sensorValue = 0;
    uppressed = true;
    res = 0;
    ScrollUP();
}
else if (sensorUP < 200) {
    sensorValue = 0;
    uppressed = false;
}

if ((sensorDN > 900) && (dnpressed == false)) {
    sensorValue = 0;
    dnpressed = true;
    res = 0;
    ScrollDN();
}
else if (sensorDN < 200) {
    sensorValue = 0;
    dnpressed = false;
}

if (analogRead(A3) > 900) {
    lcd.setCursor(0, 1);
    lcd.print(blankline);
    lcd.setCursor(0, 2);
    lcd.print(blankline);
}
```c
void UpdateLCD(void) {
    int i;

    for (i=0; i<3; i++) {
        lcd.setCursor(0, i+1); //row are 0-3, header is row zero
        stringOne = ""; //start new strike line
        if (toprow+i > c_val)
            break;
        BuildString(" ",0,3,toprow+i); //add c_val to stringOne;
        BuildString(" ",1,2,mn[toprow-1+i]); //add month to stringOne;
        BuildString("/",1,2,dy[toprow-1+i]); //add day of month to stringOne;
        BuildString("/",1,2,yr[toprow-1+i]); //add year to stringOne;
        BuildString(" ",1,2,hr[toprow-1+i]); //add hour to stringOne;
        BuildString(":",1,2,mi[toprow-1+i]); //add minute to stringOne;
        if ( pm[toprow-1+i] )
            stringOne = stringOne + "PM";
        else
            stringOne = stringOne + "AM";
        lcd.print(stringOne);
    }
}
```
void BuildString(String sym, int symcnt, int valcnt, int val) {
    // utility function: append to str # of spaces followed by val with min length of digits
    String tmpstr = "";
    int i = 0;

    while (i++ < symcnt)
        stringOne = stringOne + sym; //add symcnt symbols to end of string

    tmpstr = String(val);
    while (tmpstr.length() < valcnt)
        tmpstr = "0" + tmpstr; //add val to string with correct number of digits
    stringOne = stringOne + tmpstr;
}

void CheckAMPM(int index) {
    int hour12;
    bool PM = false;

    hour12 = hr[index];
    PM = hour12 >= 12;
    if (PM && (hour12 != 12))
        hour12 = hour12 - 12;
    if (hour12 == 0)
        hour12 = 12;
    hr[index] = hour12; //update hour array
    pm[index] = PM;
}
void ScrollUP( void ) {
    if (toprow > 1 && c_val > 1) {
        toprow--;
        UpdateLCD();
    }
}

void ScrollDN( void ) {
    if (toprow < c_val - 2 && c_val > 3) {
        toprow++;
        UpdateLCD();
    }
}

RTC Set Time Code

#include <Wire.h>
const int DS1307 = 0x68; // Address of DS1307 see data sheets
const char* days[] =
{"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday");
const char* months[] =

// Initializes all values:
byte second = 0;
byte minute = 0;
byte hour = 0;
byte weekday = 0;
byte monthday = 0;
byte month = 0;
byte year = 0;

void setup() {
    Wire.begin();
    Serial.begin(9600);
    delay(2000); // This delay allows the MCU to read the current date and time.

    Serial.print("The current date and time is: ");
    printTime();
    Serial.println("Please change to newline ending the settings on the lower right of the Serial Monitor");
    Serial.println("Would you like to set the date and time now? Y/N");

    while (!Serial.available()) delay(10);
    if (Serial.read() == 'y' || Serial.read() == 'Y')

        // This set of functions allows the user to change the date and time
        {
            Serial.read();
            setTime();
            Serial.print("The current date and time is now: ");
            printTime();
        }

    Serial.println("Thank you.");
}

// Continuous function for converting bytes to decimals and vice versa
void loop() {

}
byte decToBcd(byte val) {
    return ((val/10*16) + (val%10));
}

byte bcdToDec(byte val) {
    return ((val/16*10) + (val%16));
}

// This set of codes is allows input of data
void setTime() {
    Serial.print("Please enter the current year, 00-99. - ");
    year = 15;
    Serial.println(year);
    Serial.print("Please enter the current month, 1-12. - ");
    month = 4;
    Serial.println(months[month-1]);
    Serial.print("Please enter the current day of the month, 1-31. - ");
    monthday = 22;
    Serial.println(monthday);
    Serial.println("Please enter the current day of the week, 1-7.");
    Serial.print("1 Sun | 2 Mon | 3 Tues | 4 Weds | 5 Thu | 6 Fri | 7 Sat - ");
    weekday = 4;
    Serial.println(days[weekday-1]);
    Serial.print("Please enter the current hour in 24hr format, 0-23. - ");
    hour = 18;
    Serial.println(hour);
    Serial.print("Please enter the current minute, 0-59. - ");
    minute = 39;
    Serial.println(minute);
second = 0;
Serial.println("The data has been entered.");

// The following codes transmits the data to the RTC
Wire.beginTransmission(DS1307);
Wire.write(byte(0));
Wire.write(decToBcd(second));
Wire.write(decToBcd(minute));
Wire.write(decToBcd(hour));
Wire.write(decToBcd(weekday));
Wire.write(decToBcd(monthday));
Wire.write(decToBcd(month));
Wire.write(decToBcd(year));
Wire.write(byte(0));
Wire.endTransmission();
// Ends transmission of data

byte readByte() {
    while (!Serial.available()) delay(10);
    byte reading = 0;
    byte incomingByte = Serial.read();
    while (incomingByte != '\n') {
        if (incomingByte >= '0' && incomingByte <= '9')
            reading = reading * 10 + (incomingByte - '0');
        else;
        incomingByte = Serial.read();
    }
    Serial.flush();
return reading;
}

void printTime() {
  char buffer[3];
  const char* AMPM = 0;
  readTime();
  Serial.print(days[weekday-1]);
  Serial.print(" ");
  Serial.print(months[month-1]);
  Serial.print(" ");
  Serial.print(monthday);
  Serial.print("", 20");
  Serial.print(year);
  Serial.print(" ");
  if (hour > 12) {
    hour -= 12;
    AMPM = " PM";
  } else AMPM = " AM";
  Serial.print(hour);
  Serial.print(" :");
  sprintf(buffer, "%02d", minute);
  Serial.print(buffer);
  Serial.println(AMPM);
}

void readTime() {
}
Wire.beginTransmission(DS1307);
Wire.write(byte(0));
Wire.endTransmission();
Wire.requestFrom(DS1307, 7);
second = bcdToDec(Wire.read());
minute = bcdToDec(Wire.read());
hour = bcdToDec(Wire.read());
weekday = bcdToDec(Wire.read());
monthday = bcdToDec(Wire.read());
month = bcdToDec(Wire.read());
year = bcdToDec(Wire.read());
}

Final Schematic