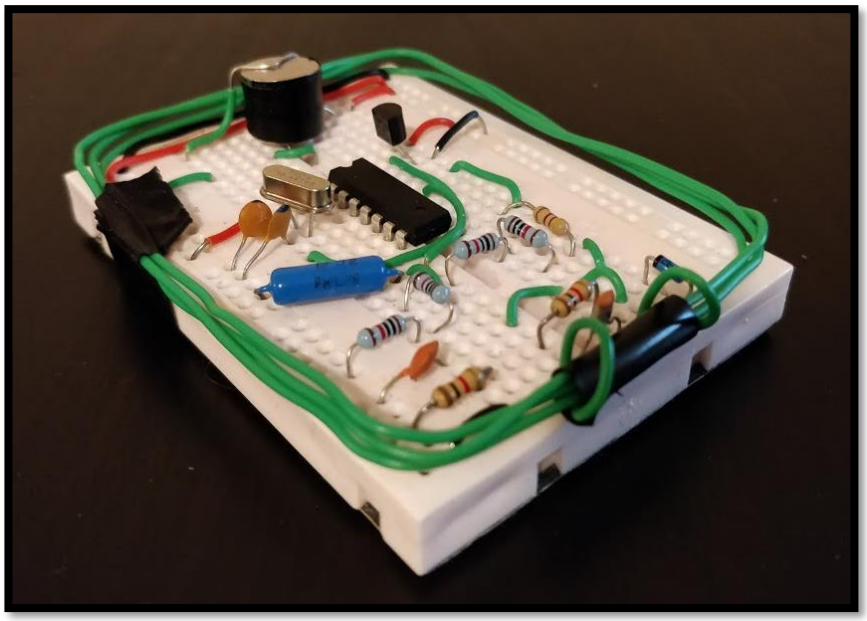
DUBLIN CITY UNIVERSITY
SCHOOL OF ELECTRONIC ENGINEERING

Passive NFC Environmental Sensor

Niall Quirke

April 2018



BACHELOR OF ENGINEERING

IN

ELECTRONIC AND COMPUTER ENGINEERING

Supervised by Dr. C. McArdle

Acknowledgements

I would like to thank my supervisor Dr Conor McArdle for his guidance, enthusiasm and commitment to this project. Thanks' are also due to Mr Billy Roarty for supporting this work to date. Finally, I would like to express my deep appreciation to Dr Derek Molloy for preparing the original Transfer Report template, from which this document is hacked.

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged, and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

I have read and understood the DCU Academic Integrity and Plagiarism at https://www4.dcu.ie/sites/default/files/policy/1%20-%20integrity_and_plagiarism_ovpaa_v3.pdf and IEEE referencing guidelines found at https://loop.dcu.ie/mod/url/view.php?id=448779.

Name: _____ Date: _____

Abstract

In this project a passive NFC temperature sensor was built as well as a basic Android application to better interface the readings. It was built using an ATtiny84 microcontroller, a basic 3 pin temperature sensor, an antenna coil and some basic circuitry components. An open source signal processing circuit and open source code were used to replicate an NFC tag while code was written to record sensor data and send it to the phone through the NFC protocol to be displayed. A 3V battery was used to power the microcontroller but a future battery-less implementation was considered in design decisions throughout. Extensive work was also done in the troubleshooting of external clock issues with the ATtiny84 and in the testing of antenna coils of different size and shape for NFC applications.

Table of Contents

Table of Figures

Chapter 1 - Introduction

The idea of the Internet of Things is that there are great benefits to having more devices connected and sharing information. Like a fridge adding things to your shopping list on your phone or your runners updating your jog performance to your health tracker app. However, there are big challenges facing IOT engineers trying to make this happen. One of the biggest challenges is that the sensors collecting all this information for us need to be cheaper, longer lasting and easier to communicate with. They need to be able to work in new areas with new design challenges.

Figure 1 - The Future of IOT [1]

The idea process for this project was started with an aim to tackle these challenges and design a sensor with enough of these qualities to be uniquely useful for a certain application. I felt the most important problems to solve with current sensors advancing into new application environments were the high expense, the short life time and the difficulty of communicating to these current sensors. Solving these engineering problems in one sensor would in turn solve many real-world problems.

1.1 A Description of the Project Problem

The project problem is that in many IOT applications it is desirable to be able to easily collect sensor data from environments without power supply or network access and there are very few options for such applications. Most sensor solutions require a battery to run but batteries introduce an expiry date and a price burden on the solution. Most sensor solutions also require internet access to communicate their readings, but network access is not available in certain areas and can be quite expensive. RFID and RFID tags are an emerging battery-less and network-less technology that allows the identification and tracking of objects under these conditions, but it is still relatively difficult to get readings from these objects and requires specialized equipment. This problem has been solved by NFC and NFC tags, technologies built on top of RFID that are now ingrained in all smartphones and so are very easily connected to by anyone. However, there is very little sensing ability available on these devices. As was mentioned, IOT is all about the great benefits from having more devices connected and sharing information. It was thought that this absence of sensing ability on NFC tags was possibly a big engineering problem with a big real-world potential. The problem this project is trying to solve is adding environmental sensing ability to an NFC tag.

1.2 A Description of the Proposed Solutions

The proposed solution is a passive NFC environmental sensor. In this project, as study into this area, a battery powered, passive NFC temperature sensor will be built, but a future battery-less implementation will be considered at every design decision. The full solution will consist of 5 main works:

- an Android app to interface the sensor information to the user,
- an antenna coil to capture the NFC signal coming from the phone,
- a signal processing circuit to translate the signal into binary,
- a microcontroller to run the NFC communication and the sensor,
- and a temperature sensor to record sensor data.

Chapter 2 – Technical Background

The technical background of this project describes any prior knowledge needed to understand or replicate this project. This knowledge can be broken down into six main sections:

- RFID and NFC
- The NFC Protocol
- Radio Frequency Tuning
- Resonant Inductive Coupling
- Signal Processing
- ATtiny84 Microcontroller

2.1 RFID and NFC

To backtrack a bit, RFID (Radio Frequency Identification) uses electromagnetic fields to wirelessly identify and track tags attached to objects. There are two main types of these trackable tags: battery-assisted and battery-less. There is a common misconception where battery-less tags are called passive tags [2]. Whether a tag is passive or not is dependent on whether it has a transmitter and can create its own channel for duplex communication or not [2]. Battery-less RFID tags are the much more common type as it is a major attraction of any RFID solution. RFID tags are commonly used across many industries including the manufacturing and transporting of goods or in the identifying of livestock or vehicles. An RFID tag consists of an antenna and a low power circuit. It is generally woken up by a reader which sends it an encoded message and it responds with its identification and other information [3].

NFC (Near Field Communication) is built on top of short range RFID technology with added features [4]. One of the main attractions of NFC is that both nodes in the communication can be readers. This is useful for transferring files between two phones for example. Commands can also be written into NFC tags to change the setting of a phone or get the phone to open a browser and go to a certain URL using the NDEF (NFC Data Exchange Format) [5]. These are commonly used in restaurants as electronic menus or in advertising with links to an online store.

2.2 The NFC Protocol

The aim of this project is to be able to hold any Android phone[1] near this passive sensor and get a reading. This is achieved by connecting the phone and sensor using the NFC protocol. The most important technical background is therefore a basic understanding of how the NFC protocol, and more specifically NFC tags, work, as the sensor will be acting as a NFC tag as far as the protocol is concerned. To use an NFC tag, you enable NFC communication in the settings on your Android phone and then hold the phone near the tag. The NFC antenna is usually built into the top of the phone around the camera so the closer this is to the tag the better. You should then hear a chime from your phone and it will do whatever the tag told it to do. So as explained in the introductions common uses include opening a browser on a website or on the credit card details page for a product or donation, changing some settings in your phone or adding a new contact. This section will break down what technically happens in this handshake between the phone and tag.

An NFC tag is made up of 3 main parts, an antenna, an analogue circuit and a microcontroller chip. When NFC communication is enabled on your phone it is periodically casting out a polling signal looking for nearby NFC devices such as other Android phones, an Android pay terminal, an NFC speaker or an NFC tag. When the antenna on the tag gets a polling message at 13.56MHz (the protocol standard) a circuit processes this signal and feeds a binary message to the microcontroller using Miller encoding. The binary being sent from the phone to the tag is always Miller encoded while the binary sent from the tag to the phone is always Manchester encoded. At the same time as picking up the message the antenna is also inductively powering the microcontroller so there is no need for a battery, greater detail to follow [2.4 Resonant Inductive Coupling]. The microcontroller then responds to the phone by load modulating the end of the signal sent from the phone. Load modulating is the process of varying a resistor inside the microcontroller to draw varying degrees of current from the phone in a Manchester encoded binary pattern. This first message provides the phone with some info about the tag like which type of tag it is, it's ID, etc. The phone then does a quick scan to make sure there is no other tag interfering in the field and then starts a communication channel with the tag [6]. The protocol will discussed in greater detail as the code in explained in [

[1] Apple have locked the NFC hardware on iPhones to in-house Apple applications such as Apple Pay making them incompatible with the device in this project. See 3.2.1 NFC: Android Vs iPhone

Chapter 7 – NFC Protocol Code] but for the full details refer to the protocol documents [7] [8].

2.3 Radio Frequency Tuning

The first job of an NFC tag is to pick up the signal being polled by the phone. For the device's antenna to be able to pick up this signal it must be tuned or optimized for a certain frequency of communication. This frequency is 13.56MHz as defined in the NFC protocol [7]. Making a basic antenna that is tuned to this frequency involves a coil of wire and a capacitor.

When a coil of wire and a capacitor are in series in a powered circuit they resonate the current at a certain frequency. This is known as the resonant frequency. A resonating circuit's current is very easily affected by external electromagnetic waves at its resonant frequency. The way the current is affected can be detected and translated back to binary that a computer can understand and work with. This is the basic principal behind all modern wireless communication from WIFI to radios.

As part of this project an antenna was designed for the circuit and tuned to 13.56MHz to communicate through NFC to the phone. This was done by making a coil of wire with a certain inductance and using a capacitor with a certain capacitance that would cause the current to resonate at that frequency. The inductance of a coil of wire depends on three factors: the number of loops it has, the diameter of the loops, the diameter of the wire and relative permeability of the material. This can be expressed in the following equation:

$$L = (N^2 * \mu * A) / h$$

Where L = inductance, N = number of loops, μ = permeability, A = area of coil

The equation for the resonant frequency of this circuit and all its dependants can be expressed:

$$F = 13.56 * 10^6 = 1 / (2\pi * \sqrt{(L * C)})$$

Where F = frequency, L = inductance, C = capacitance

We used a 56pF capacitor in this project so to resonate at 13.56MHz that left us needing a coil with an inductance of:

$$13.56*10^{6} = 1 / (2\pi * \sqrt{(L * 56 * 10^{-12})})$$

$$(2\pi * 13.56 * 10^{6})^{2} = 1 / (L * 56 * 10^{-12})$$

$$L = 1 / (4 * 56 * 10^{-12} * \pi^{2} * (13.56 * 10^{6})^{2})$$

$$L = 0.00000386 = 3.86\mu H$$

Now that we know what inductance we need we just need to give the antenna the right number of loops and the right diameter etc. to make up this inductance value using the first equation. There are many calculators online to help with calculations and explaining the maths [9].

2.4 Resonant Inductive Coupling

As explained earlier not only does the NFC tag receive data from the phone but it also receives power. This is done by inductive coupling between the antenna of the phone and the antenna of the tag. “Unlike traditional antennas, an NFC antenna utilizes the inductive coupling of magnetic field for power and data transfer instead of electromagnetic radiation” [10]. In this way the coupling acts as an air-core transformer.

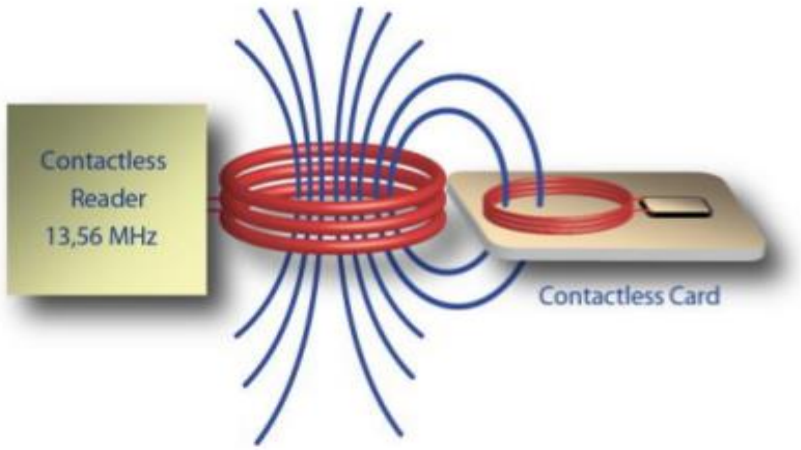


Figure 2 - NFC Inductive Coupling [11]

The phone flows a current through its antenna coil setting up a magnetic field. When the tag comes within range of this field the magnetic force around the tags antenna coil causes a current to flow. This is the basis of Faraday's law which is widely covered for example [12].This current is then used to power and send data to the tag. The advantage of this connection is that the tag can be completely battery-less.

2.5 Signal Processing

The second job of the device, after tuning in to the signal being sent, is to convert this alternating current into binary message the microcontroller can understand. Therefore, there is a circuit between the antenna and the microcontroller. This is most effectively done with some analogue circuit components as the frequency is too fast for a digital device to sample. There are number of different steps involved in this circuit depending on the signal being sent, the device receiving the data and how the binary is encoded. The details of each step of the processing of the signal in this project will be discussed later [Chapter 5 – Processing the Signal].

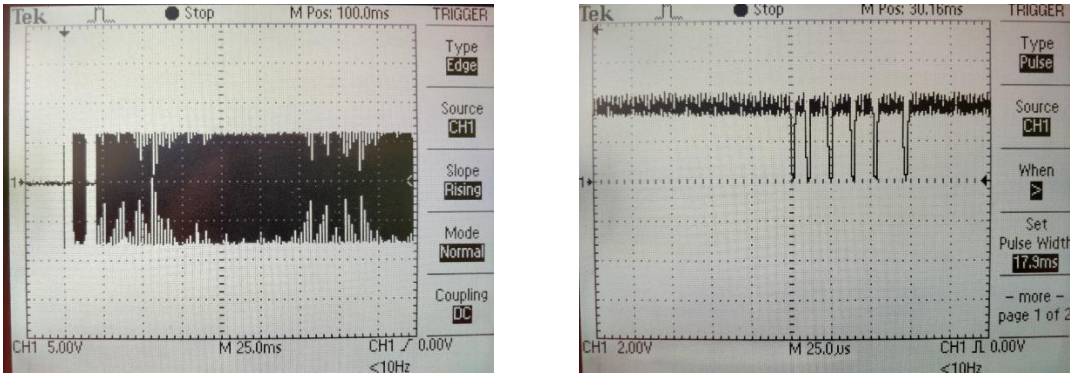


Figure 3 - Signal Before and After Processing

2.6 ATtiny84 Microcontroller

The final piece of technical background knowledge required for this project is a basic understanding of microcontrollers and how they work. The microcontroller is the heart of this project, it is where all the main functionality is performed and where most of the work and troubleshooting is required. The role of the microcontroller is to read in the encoded binary from the analogue circuit, construct and send responses in the form of a full NFC handshake, read in an analogue value from the sensor and send it to the phone.

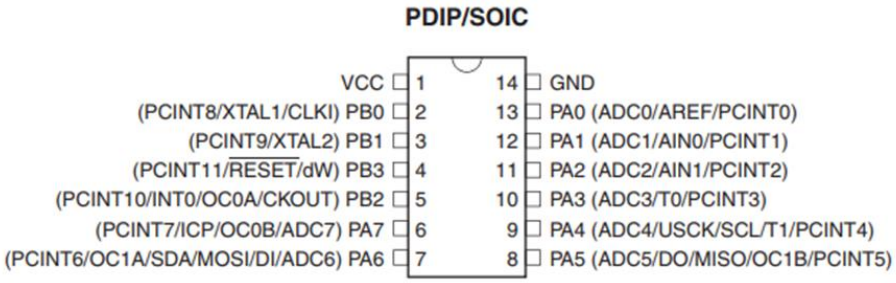


Figure 4 - ATtiny84 [13]

Microcontrollers are small computers on a single integrated circuit, usually used for dedicated, embedded applications. They consist of a CPU, RAM, ROM and I/O ports. The ATtiny84 microcontroller was chosen for this project because it is relatively small, it has 14 pins, it has relatively low power consumption and it is easily programmed using an Arduino.

Chapter 3 – Ethical Considerations

The following chapter discusses some of the relevant ethical considerations for this research. It is very important to include an ethical consideration for any complete research or innovation to keep the world moving in a positive and caring direction. As a leading example the IEEE Code of Ethics commits "to hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices, and to disclose promptly factors that might endanger the public or the environment" [14]. The innovation involved in this paper, like any other, is not without its ethical concerns. The main concerns can be broken into three main factors: Health and Safety; Data Protection and Environmental Impact.

3.1 Health and Safety

The first ethical consideration is to do with the health and safety of the device in its given application environment. Sensors have such diverse application environments safety concerns vary greatly from application to application. I will first discuss the safety concerns in relation to the circuit design involved in the making of the sensor and secondly the safety concerns in relation to specific application environments of some possible use cases for the passive NFC sensor.

3.1.1 Safety Consideration for Circuit Design

The two main parties for a circuit designer to consider are the user and the service personnel. The user presumably having no knowledge of the risks but a low-level exposure to the circuit, usually due to some cover. While the service personnel are presumed to have some knowledge of the risks but a higher exposure to the circuit. The main safety threats circuit designers should be concerned with are:

- electric shock
- energy hazards
- fire
- heat related hazards and
- mechanical [15]

Additionally, it is important for circuit designers to not only consider the normal operating conditions but to also take things going wrong into account.

Electric shock can result in anything from a twitch of a limb to death depending on voltage levels and has been shown statistically to be one of the main causes of death in the home [16] and workplace [17]. For voltages too low to cause an electric shock but with a high VA potential, components from the circuit can still cause a burn especially when they meet another metal object like a ring. This or other occurrences in the circuit can then lead to a fire spreading as was the case with the famous Samsung phone recall recently. Even under normal operating conditions components in the circuit can get hot or if there are any sharp edges or moving parts on the circuit it may not be suitable to touch [15].

As with all electrical standards the standards outlining minimum safety requirements varied greatly from country to country and industry to industry in the past but has become much more international in recent times. The current main governing standard now is the 2016 version of the IEC 62477 defined as "Safety requirements for power electronic converter systems and equipment - Part 1: General" [18].

3.1.2 Safety Consideration for Passive NFC Sensor Application Environments

To talk about the safety of sensors themselves without mentioning the application environment would not be much use. The environments vary hugely, and every environment comes with its own issues. A passive NFC sensor could have many applications, but I have picked a few that I am specifically aiming for with this project. These have been chosen because they highlight the unique selling points of this type of sensor and so are the most likely place for the sensor to be implemented. These applications are:

- measuring air pressure inside a tyre
- measuring gas levels inside a tank and
- measuring temperature of food inside a can

I will note the main safety concerns of each which I hope will in turn give an idea of the safety concerns to be considered in any environment for this type of sensor.

Obviously, there isn't too many safety concerns inside a car or bike tyre but this environment alongside the others is a good example of how varied the level of safety required can be with

this type of sensor. The only concerns in this environment are the before mentioned circuit design concerns where an overheating might set fire to the rubber of the tyre. This is very unlikely though considering the low voltage levels the circuit is working at.

The second environment should be given a lot more consideration. Depending on the gas involved, inside this container could be a very flammable environment and if the circuit were to overheat or spark at all there would probably be an explosion. For this application a passive NFC gas or air-pressure sensor might be used, to measure level of a certain gas or to measure how much is left in a compressed container, respectively. For either of these cases the type of gas would be very important but as an example I'll talk about a classic butane gas tank popularly used for cooking. A cheap and easy way of checking how much is left would be to stick a passive NFC air-pressure sensor on the inside at a marked point for a phone to touch and get a measure of the pressure. Butane has a Lower Explosive Limit (LEL) of 1.8 meaning if there is as much as 1.8% of butane in the air and an ignition source is present it will burn [19]. An overheated or sparking circuit inside a compressed cylinder of butane could have devastating effects. Again, this is quite unlikely at the low-voltage levels but due to the potentially deadly result of this event it should definitely be taken into design considerations.

The third environment is another completely different situation. This time safety concern is entirely to do with the covering material of the circuit and sensor. The material couldn't affect the temperature measurement but if it was not taken into consideration at design stage it quite likely that the food would be contaminated. The main regulatory standards to adhere to in this situation would be Framework Regulation (EC) No. 1935/2004 in Europe enforced by the EU and 21 CFR (Code of Federal Regulations) 174-179 in the US enforced by the FDA [20].

3.2 Data Protection

Data Protection is a big conversation in the NFC space. Being a relatively new wireless communication tool, it is bound to face a lot of hackers and hacker technology, especially when it is becoming particularly popular as a payment tool. An interesting learning curve in this project was how easy it was to pick up the conversation between the phone and an NFC device. With just a coil, a rectifying circuit and an oscillator I could pick up and recognise the byte messages between by phone and my credit card! My phone could only get so far in the conversation without certain keys, but you can see the potential for a hacker with much more

sophisticated equipment. Having said this, there is a reason it is becoming popular for wireless payments. Requiring such a near field to communicate is actually a big advantage in many ways and does narrow down the risk of large-scale, international attacks compared to some methods of communication.

In this project the data at risk is the value that the sensor is sensing and sending back to an android phone with the app included in this project. For most applications that have been considered for this type of sensor in this project this is not a problem because it is not particularly sensitive data i.e. the air-pressure of a tyre or the amount of gas left in a cylinder. However, sensors like this one have very wide and varied application and it might find a use case where data protection is required. In this case it would be important to have a more powerful chip capable of encryption.

3.2.1 NFC: Android Vs iPhone

It is for these very reasons that I'm using Android phones throughout this project. iPhones do indeed have NFC hardware in them and they are used for applications such as Apple Pay. However, Apple do not open the NFC facilities on the phone to Appstore developers. The NFC hardware on iPhones can only be used by in-house Apple apps and is locked from all other apps. Apple says opening NFC hardware to 3rd party developers is too risky and they have a much more secure set up [21].

3.3 Environmental Impact

The final ethical consideration for this project is to do with the environmental impact of widespread use of a passive NFC sensor. The type of impact this would have on the environment would depend on 2 factors: the materials used and the quality of the inductive connection between the phone and the sensor.

The materials used is an obvious one and is an environmental concern for every hardware project. The only hardware materials required for this project are a circuit board, an antenna and a sensor of whatever kind. However, a lot of application for these types of devices mean they are used in a mass scale and are possibly quite disposable because one of the key unique selling points of the passive NFC sensor is that it is so cheap to use. If this sensor was to be used on a large scale and they were quite disposable there should be a lot thought, put into

stream-lining the materials. Because this is only a beta version that was not a major concern in this project.

A less obvious note on the environmental side is the concern about inductive charging. Inductive charging is becoming increasingly popular as a technology with particular interest from the phone, computer and automotive industries. It would be very handy if your laptop charged away as you use it on a desk or if your car could charge wirelessly as it sat in a parking space (particularly a self-driving car that might struggle with a charging wire) but is this really what we want for the world? Inductive charging is a very inefficient transfer of energy. The impact of inductive charging of this project is quite small but it is an interesting conversation if it was to become widespread.

Chapter 4 – Capturing the Signal

4.1 Initial Testing

The first step in this project was to capture and process the signal coming from the phone. According to the NFC protocol the phone is periodically sending out a polling message searching for NFC devices to connect with. The best place to start this project is trying to detect this signal and make some sense of it. One of the earliest experiments done to detect for the magnetic field was to connect an LED to either end of a coil of wire and put it up against the antenna of the phone while NFC was enabled.

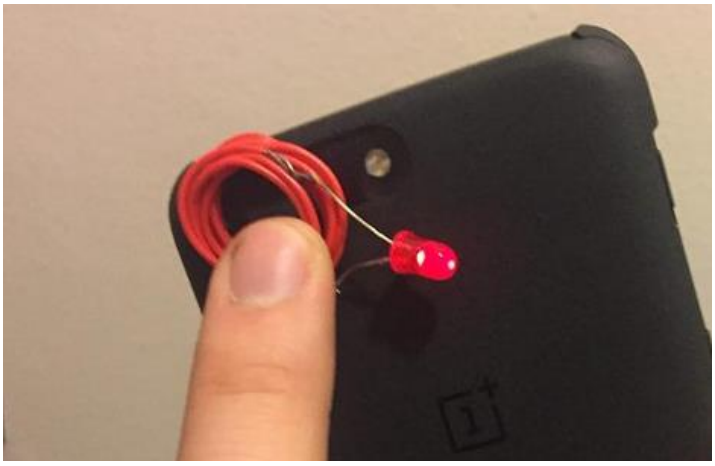


Figure 5 - LED Lighting from Inductive Coupling

This made for a very interesting first result. The LED lit up quite well every 2 seconds or so. What's happening here is the antenna in the NFC antenna in the phone is applied a current and the current going through this coil creates a magnetic field. When the coil of wire with the LED enters the magnetic field, the forces cause a current to flow in the coil which light up the LED. This is called inductive coupling and was described in more detail earlier [2.4 Resonant Inductive Coupling]. Usually for inductive coupling to be effective the coils must be tuned to resonate at the same frequency. If they are not tuned to the same frequency, there is a much greater power loss. However, this experiment showed that the signal the phone sends out must be strong enough to overcome this in this situation.

This simple experiment was also useful in getting an idea how NFC is used on phones. For example, it was noted that the LED would not light if the phone was locked. This is for security reasons, no one should be able to contact your phone through NFC while it's locked or you're not using it. It was noted that while most apps were open nothing changed but when

certain apps were open the LED would stop lighting. This is presumably to save power consumption while using some power intensive apps. Finally, it was also noted that when an NFC tag was placed near the phone the LED would stop flashing and light continuously until the tag was taken away again. This was demonstrating how the feature of the protocol that says the phones stays connected to a tag until it has left the field.

4.2 Antenna Design

While this last coil of wire was fine for detecting the signal coming from the phone, something more sophisticated is required to get the most out this signal. Specifically, the next step is to decide on parameters for a resonating antenna coil that could get the maximum power transfer from the phone.

As described in the technical background [2.3 Radio Frequency Tuning], the relevant parameters include number of loops, coil area, permeability and wire diameter. It is desirable in this project to have the coil of wire roughly around the same size as the phones antenna (the reason for this is discussed under inductive powering). Therefore, the first demo coil was chosen to be circular shaped and 50mm wide with 4 loops and this worked quite well. However, it was difficult to find a lot of information on what size or shape antenna coil would be best for NFC applications. For that reason and because the antenna coil had such an important role in the success of this project it was decided to do some testing on the different characteristics of the coil and measure performance in relation to NFC applications. The rest of the antenna design will be based around this testing in chapter 10.

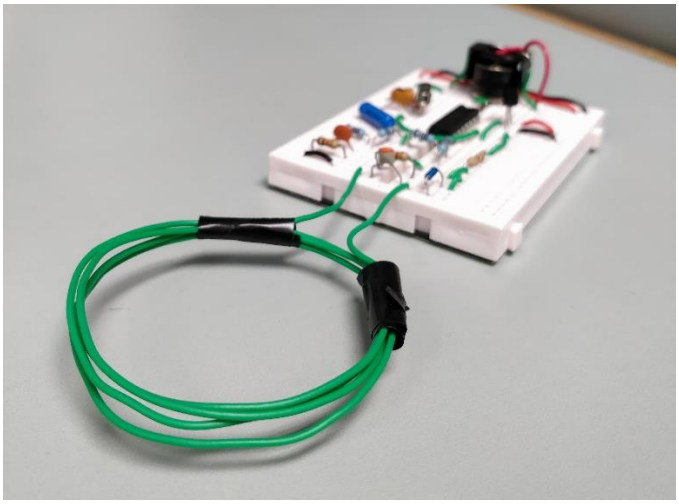


Figure 6 - Initial Demo Antenna Coil

Chapter 5 – Processing the Signal

Once we have captured the signal sent out from the phone we need to process this signal into binary the microcontroller can understand. This is done with an analogue circuit between the antenna and the microcontroller. For this project, an existing circuit design was sourced from this open source project [22].

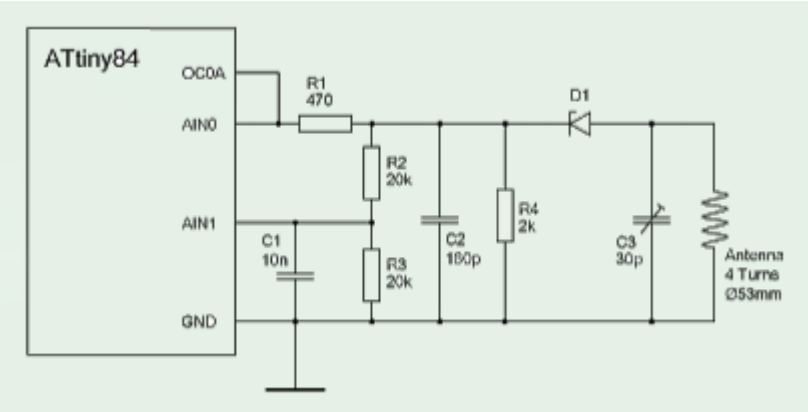


Figure 7 - Open Source Circuit Design [22]

We will now break down each step of this circuit and explain why it's there.

5.1 Rectifier

The first thing to do with the signal is rectify it. The alternating current in the antenna needs to be converted into a direct current so it can be fed into the microcontroller. Rectifying a signal is the process of filtering out the negative amplitude and being left with only the positive part of the signal. This is done using a diode because a diode will only allow the current to flow in one direction. When a diode is put after an AC wave in a circuit the result is what's called a half-rectified wave because half of the wave is lost to the filtering process. However, if a capacitor is linked to ground just after the diode it will charge up while the diode is letting current flow and discharge when the diode is stopping current flowing which result in a fully rectified wave [23].

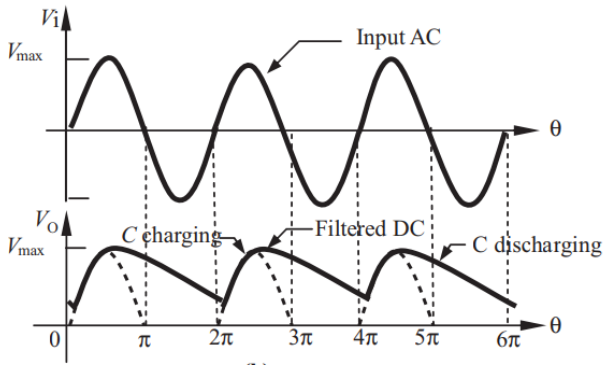


Figure 8 - Rectifier Waves [23]

In the circuit in this project I used a Schottky diode because it has a particularly low forward voltage drop and a fast switching action. The most important parameter in this circuit is power consumption. It must be kept to a minimum for a passive NFC tag implementation. Below the input signal and output signal of the diode in the projects circuit are shown, as seen on an oscilloscope.

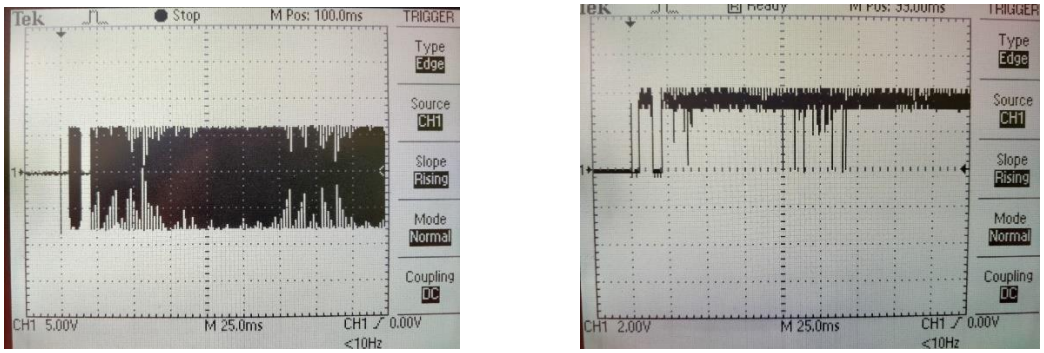


Figure 9 - Diode Input and Output Respectively

It can be seen from these screens that the diode and capacitor have completely filtered out the negative amplitude and there is a fully positive signal left.

5.2 Envelope Detector

When demodulating a signal, it is only the amplitude of the signal that's important and not the carrier frequency. The amplitude of the signal is much easier to detect when this carrier frequency is filtered out. This is what the envelope detector is for.

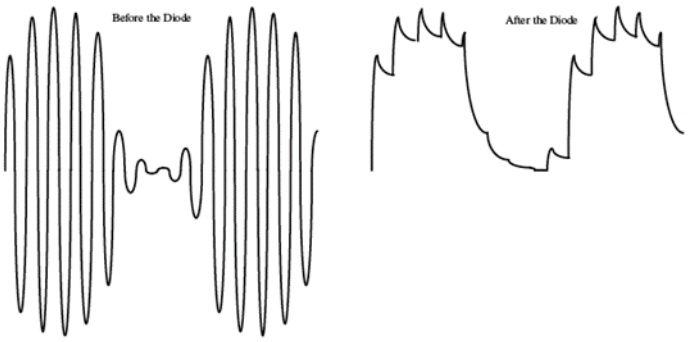


Figure 10 - Envelope Detector Input and Output Respectively [24]

An envelope detector is comprised of a diode, a capacitor and a resistor configured like a half rectifier circuit and a capacitor to ground in series. The capacitor and the resistor act as a low pass filter, filtering out the high frequencies while the diode filters out the negative amplitude. In this project resistor and capacitance values were calculated in order to filter out the 13.56MHz carrier frequency. Below the results of the envelope detector can be seen as shown on the oscillator.

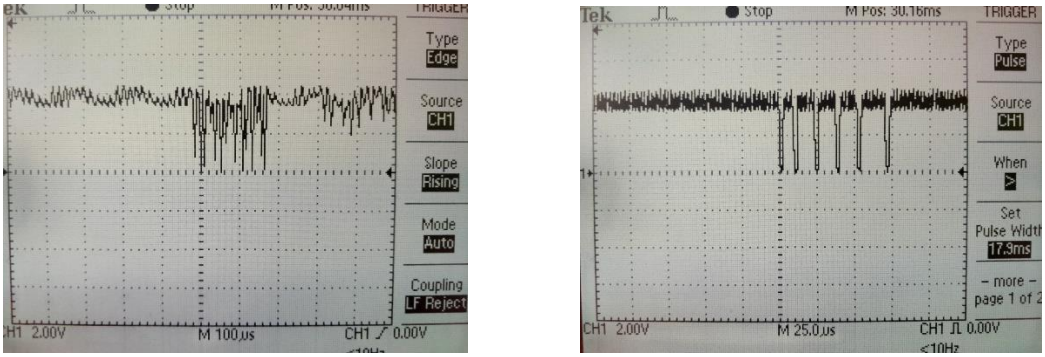


Figure 11 - Project Envelope Detector Input and Output

The carrier frequency was filtered out and the resulting signal has a much clearer amplitude. This signal is much easier detected by the microcontroller.

5.3 Comparator

The microcontroller determines the binary information as interrupts at certain points along its clock cycles. These interrupts in the NFC chip are trigger by a comparator in the microcontroller. The comparator compares the analogue input coming in on 2 different pins and signals an interrupt if they differ. Seen below are the inputs on the 2 different pins.

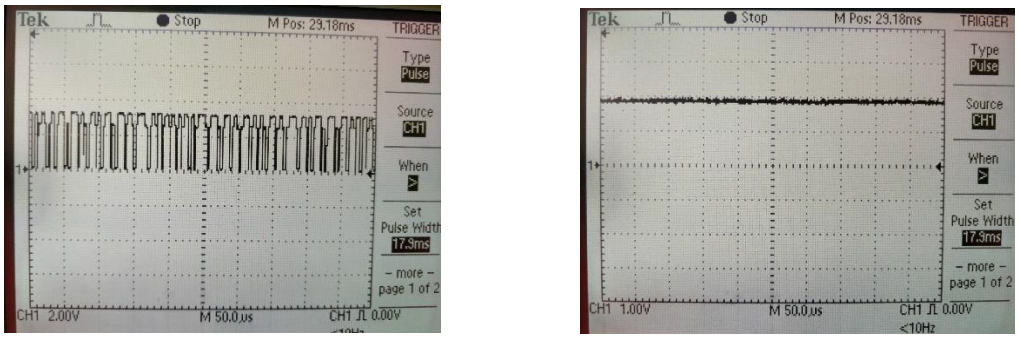


Figure 12 - Message Input and Comparator Input

The first signal shown is the message input. This contains the binary information that signal the 1 and 0 interrupts. It comes out the envelope detector, through a resistor to pull more current from the phone and into the AIN0 pin on the ATtiny84 microcontroller. The second signal needs to be created as a completely high signal, so the other signal can signal interrupts when compared against it. For this reason, the signal coming from out of the envelope detector is also fed into a very strong low pass filter to take all the dips put of it before it is fed into the comparator AIN0 pins on the ATtiny84 microcontroller.

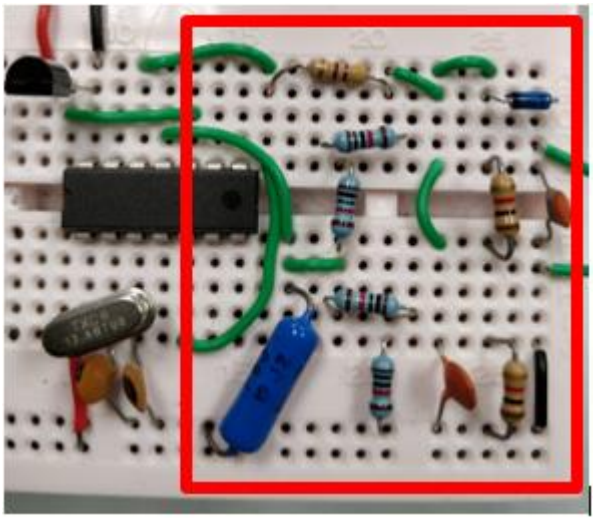


Figure 13 - Signal Processing Circuit

Chapter 6 – Programming the NFC Chip

Once we have the signal processed in a way the microcontroller can understand, the next step is to program the microcontroller to interpret the message and construct a response. The task of programming the microcontroller itself was not as straight forward as it first seemed though. Most NFC tags consist of very complicated and specific circuits and computer chips. The challenge of this project was to break down the bare bones of the NFC protocol and try to run it on a simple, cheap, generic microcontroller, keeping the power consumption as light as possible so it could run a sensor as well. For this to be possible, when programming the ATtiny84 microcontroller it was important to get the very most out of. This included using a very low-level compiler and programmer, running it on a faster external clock, using all 14 pins and even configuring firmware fuses in the chip to run differently. These are the types of things we will talk about in this chapter.

6.1 In-System Programmer

The ATtiny84 microcontroller and many microcontrollers like it have no USB ports or internet connectivity so the only way to program them is through a hardware programming device. There are many built specifically for the ATtiny84 chip and the ATtiny family like this one.

Figure 14 - In-System Programmer for the ATtiny85

In this project however it was decided to use the Arduino as the ISP just because they were easily available. This can be done by uploading the “Arduino as an ISP” example sketch to the Arduino, adding the AVR library to your Arduino IDE and compiling further programs for the ATTiny board in settings. The board will also need to be connected to the Arduino using the following pins configuration. (a more detailed description can be found here [25])

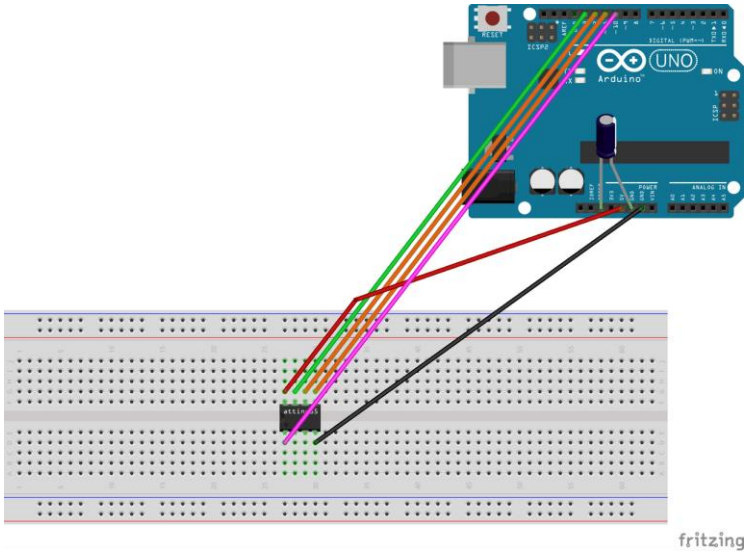


Figure 15 - Arduino as an ISP Configuration

The ATtiny84 microcontroller is designed to do almost everything an Arduino can do but in a more compact and cheaper chip. The main thing it's missing compared to an Arduino is the UART (Universal Asynchronous Receiver-Transmitter) which means it's difficult to get a serial output from. This meant a lot of troubleshooting in this project was done using LEDs rather than a serial monitor. It also has 8KB of program memory space, 20MIPS CPU speed, 512 bytes of SRAM and 512 bytes of EEPROM. It is very similar to the more popular ATtiny85 except it has 6 more pins and a 16bit counter which were both essential for this project [13].

6.2 Compiler

Initially the Arduino IDE and compilers were used for compiling and uploading the code. This worked quite well for an LED blink test and some other testing codes. However, as the clock speed became an issue and it was required to run the microcontroller at 13.56MHz the Arduino IDE was unable to compile and upload the code to work at this speed. The AVR library for the Arduino IDE has limited options for clock speed including only 8MHz, 16MHz and 20MHz for an external clock. Even when it was defined in the code for the clock to run at 13.56MHz if the chip was not compiled to run at this speed it still only ran at whatever speed was set by the Arduino IDE.

To get the code compiled at 13.56MHz the lower-level AVRdude compiler was used. This is the program inside the Arduino IDE that deals with all AVR boards. When it is called from command line it is free of some constraints the Arduino IDE puts it under. When using

AVRdude the code first had to be compiled from a main.c file to a main.bin file. Then it had to be converted into a main.hex hex file and then it was uploaded onto the chip through the Arduino device. Because the Aruino IDE was no longer automatically including libraries it had to be made sure that all libraries needed to be included in the code were in the same file as the main.c being compiled. This is included all the AVR libraries. The AVRdude command used to compile a main.c program into a main.bin binary file was as follows:

```
avr-gcc -Wall -g -Os -mmcu=attiny84 -o main.bin main.c
```

Here the avr-gcc is calling the gcc compiler function. The mmcu is the device. You also give the name of the file and the name of the compiled file to be as arguments. The AVRdude command used to convert a binary main.bin into a hex main.hex file was as follows:

```
avr-objcopy -j .text -j .data -O ihex main.bin main.hex
```

The AVRdude command used to upload the hex file onto the microcontroller was as follows:

```
avrdude -p attiny84 -c stk500v1 -U flash:w:main.hex:i -F -P com1 -b 19200
```

The device, programmer (Arduino acts as a stk500v1), the hex file, the port and the baud rate are given as command line arguments.

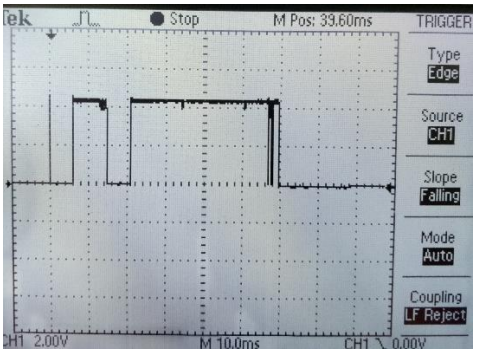
```
avrdude: please define PAGEL and BS2 signals in the configuration file for part ATtiny84
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.05s

avrdude: Device signature = 0x1e930c
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: please define PAGEL and BS2 signals in the configuration file for part ATtiny84
avrdude: reading input file "main.hex"
avrdude: writing flash (1750 bytes):

Writing | ################################################## | 100% 3.26s

avrdude: 1750 bytes of flash written
avrdude: verifying flash memory against main.hex:
avrdude: load data flash data from input file main.hex:
avrdude: input file main.hex contains 1750 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 1.96s

avrdude: verifying ...
avrdude: 1750 bytes of flash verified

avrdude: safemode: Fuses OK

avrdude done.  Thank you.
```

Figure 16 - AVRdude Compiler

6.3 Clock Speed

6.3.1 Troubleshooting Clock Speed

The most time-consuming troubleshooting in this project was the troubleshooting to do with the clock speed of the microcontroller. It is a crucial part of the communication, but it is also a very delicate process and must be configured with great precision. For the tag to be able to communicate with the phone it must "piggy back" its message on the back of the signal sent from the phone as per the NFC protocol [7]. It must do this in sync with the phones clock speed and as a subcarrier on the initial signal. If this subcarrier return message is off sync with the phones clock it won't make any sense to the phone. If it is only slightly off sync with the phones clock the phone and the tag might get one or two messages across but as the communication goes with the clocks running at slightly different speeds they will only be getting more and more off sync and the communication eventually breaks down. This is something to be wary of when troubleshooting. Initially in this project before the microcontroller had been programmed to construct a response for the phone the first picture below was be seen on the oscilloscope as the message being received at the microcontrollers analogue input pin.



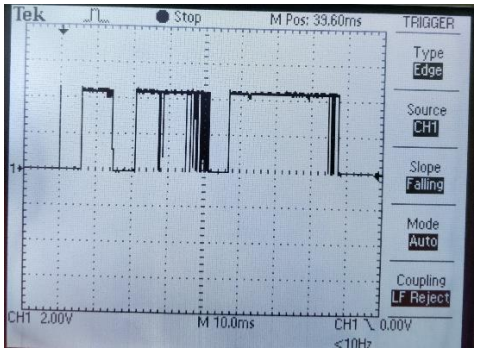


Figure 17 - Phone Polling Message and NFC Handshake Breakdown

This is the polling message the phone is constantly sending out as explained earlier. After the clocks had been set and the microcontroller had been programmed to construct a response, the second picture was seen on the oscilloscope. Here we can see the initial polling message, a reply from the tag "piggy backing" on the signal (at a much higher frequency), then another reply from the phone, maybe some response from the tag but then the phone would drop the signal and the communication would break down. For a very long time it was thought that there was a problem with the code in the tag or with the way the NFC protocol was implemented that was causing the NFC handshake between the phone and the tag to break

down. It was thought that the tag must be sending the wrong message, or the message was corrupted.

After a lot of troubleshooting it was realized that this was a clock speed issue. This was confirmed by writing test code that would blink an LED exactly every second and then starting a stop watch beside the LED (ATtiny84 has no UART for serial monitor troubleshooting). If you how counted how many times the LED blinked compared to the stop watch you could see that the LED was blinking slightly slower than the stop watch confirming the clock speed to be off sync. The clocks of the phone and tag were so close to being in sync that they managed to send over and back a few messages before the clocks went too out of sync and the communication broke down.

6.3.2 Setting ATtiny84 Fuses for an External Clock

It turns out the internal clock in an ATtiny84 runs at a maximum of 8MHz. For this project the microcontroller is required to run at 13.56MHz to be able to keep in sync with the NFC protocol. Fortunately, it is possible for the ATtiny84 to run clock speeds up to 20MHz, but it requires an external clock [13].

There are several steps to take when configuring the ATtiny84 to run on an external clock. The first of these is to configure its fuses accordingly. The ATtiny84 fuses are comparable to the BIOS configurations of a PC. The microcontroller checks them as it turns on to see how it should be set up and how it should run, whether it has a bootloader, what clock speed and voltage it will be running at, etc. What each fuse is for is explained in the data sheet but the best way to figure out about them is to have a look at a AVR fuse calculator like this one [26]. To read and set the fuses you use command line arguments into a software called AVRdude. The command arguments include three hex numbers that represent 14 fuses which are set to 1 or 0. It is quite complicated to work out the hex numbers for your desired configuration, so this calculator makes it much easier. You simply tick the boxes you want, and the calculator will give you the hex number you need. The command used to read the fuses was as follows:

```
avrdude.exe -c stk500v1 -p attiny84 -P com1 -U lfuse:r:-:i -v -C
"C:\Program Files (x86)\Arduino\hardware\tools\avr\etc\avrdude.conf" -b
19200
```

Avrdude.exe calls the program. “stk500v1” is the programmer used, apparently the Arduino as an ISP works exactly like a stk500v1 so you use that one. The attiny4 is the device. The com1 is which USB port were using. “lfuse:r:-:I” means we want to read the values. Next is the file of the avrdude.conf file which the programmer uses for this. Next is the baud rate. The command used in this project to set the fuses was as follows:

```
avrdude.exe -c stk500v1 -p attiny84 -P com1 -U lfuse:w:0xce:m -U
hfuse:w:0xdf:m -U efuse:w:0xff:m -v -C "C:\Program Files
(x86)\Arduino\hardware\tools\avr\etc\avrdude.conf" -b 19200
```

A lot of the attributes here as the same as before except the main difference is we have an lfuse (low fuse), hfuse (high fuse) and an effuse (used for defining id these fuses should be rewritable. You set these three attributes with your desired hex values. For example, here the values are set to CE, DF and FF.

6.3.3 Connecting the External Clock to the Microcontroller

Once the fuses are set on the ATtiny84 you will no longer be able to program it without an external clock connected to it. You can’t even read the fuses again without the external clock. This is because the device checks the fuses at boot up and from that moment its depending on an external clock to be able to run. As explained on the ATiny84 data sheet the external clock must be connected to the XTAL1 and XTAL2 pins as shown below.

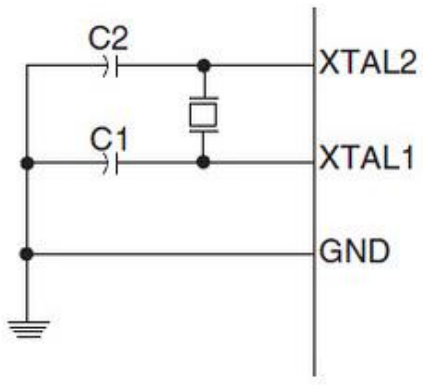


Figure 18 - Connecting a Crystal Oscillator to the ATtiny84 [13]

You must also connect either leg of the oscillator to a capacitor and then to ground. The C1 and C2 capacitance values are always equal and dependant on the frequency of the oscillator. This information can also be found in a table in the data sheet.

CKSEL3:1	Frequency Range (MHz)	Recommended C1 and C2 Value (pF)
100[1]	0.4 - 0.9	–
101	0.9 - 3.0	12 - 22
110	3.0 - 8.0	12 - 22
111	8.0 -	12 - 22

Figure 19 - Capacitance Values for an External Clock on the ATtiny [13]

For this project 2 22pF capacitors were used as the frequency range is greater than 8MHz and they worked fine.

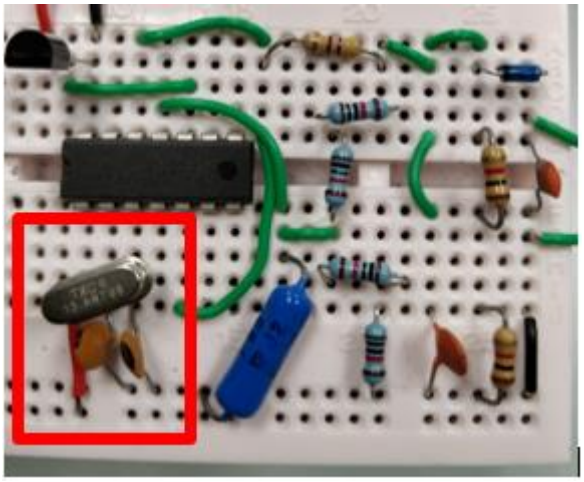


Figure 20 - The Projects External Clock

Chapter 7 – NFC Protocol Code

Now that all the hardware has been set up and the microcontroller is in position to take in and send back information, the next step is to write the software that will enable the microcontroller to understand and construct NFC messages. The code for this project is based on code from an open source project online found here [22]. This code will now be explained in as much detail as necessary along with any changes made.

It is worth noting that even though this code was not written as part of this project, this code did not work as it was first found. There was a lot of troubleshooting to be done to get this code working on the compiler and programmer being used in this project. There was also a lot of work done with this code to integrate the sensor with the NFC protocol. To troubleshoot for the compiler and programmer and to integrate the sensor a thorough understanding of the code was required. Including how every function and almost every line worked. Coming into this project with very little experience in the C programming language and no experience programming microcontrollers this was a very difficult task. This understanding of the code and the microcontroller hardware it deals with were a huge part of the time and effort put into this project.

7.1 Open Source

7.1.1 Licence

At the very start of this open source code that was downloaded from a link Github was the following message:

```
/*****************************************************************************
Written and Copyright (C) by Nicolas Kruse

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*****************************************************************************/
```

This is a common start to any open source code you get online. It explains the licencing details. These determine who can use the code, what they can do with it and what conditions must be met if the code is being used. Licencing is a critical part of the open source community it allows contributors to get credit for their work and any other benefits that might come from their work being widely distributed. It is so important because if contributors don't get credit or benefits from their work they will stop contributing and the whole community falls apart.

This code is licenced under the Apache Licence Version 2 as mentioned above. This open source licence and other open source licences are often attached to these works to allow others in the community to use them. When this licence is attached to a copyrighted work it allows anyone to use the code for any purpose, to distribute it, to modify it and to distribute modified versions of it. These actions are usually restricted by copyright laws but attaching this licence lifts these restrictions under certain conditions. In the case for the Apache licence Version 2 these restrictions include keeping the copyright notice, keeping the licence notice, stating any changes made clearly and if the source includes a "Notice" file this must also be kept in the distributed copy. Another important point the licence makes is that the owner of the code cannot be held liable for any damages caused by the distribution. The full details of the licence can be found here [27].

All these rules and restrictions were carefully adhered to in this project. This mainly consist of keeping the notices of the copyright and licencing information at the top of the code and clearly stating where any changes were made. There is also a responsibility when modifying the code to make it clear why the changes were made and how they work. If this code is to be redistributed to continue contributions to the open source community it must be easily understood to the next person. It is only fair you to give back to the community when you have benefited from it.

```
//>>>Passive NFC Sensor Code>>>
void ADCinit()
```

7.1.2 Main

The main code is quite simple for the NFC tag implementation. First the copyright and licencing information are displayed as explained. Next there are 3 libraries included:

```
#include <avr/io.h>
#include <util/delay.h>
#include "nfcemulator.h"
```

The avr/io library includes the basic input and output definitions for all the avr devices. This library makes it possible to recognise the different attributes of the ATtiny84 in the code. The util/delay library provides the functionality to delay or sleep the microcontroller for a certain length of time. The nfcemulator library is custom made for this NFC tag implementation. It provides all the variables and functions a user of this software might need visible or might want to change. These includes defining the clock speed, the NFC commands and the main functions of the software "setupNFCEmulator" and "checkForNFCReader".

Next is the definition of the NFC tag storage.

```
uint8_t tagStorage[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00,… etc.
```

The storage is just an array of hex values. However, NFC tags have a certain structure of memory as defined by the NFC Forum. There are 2 different types of memory structure for the NFC type 2 tag, static and dynamic. The static memory structure is a bare bones structure for light weight tags with 64 bytes of memory. Any tag that has over 64 bytes of memory, like the one in this project, has a dynamic memory structure as seen below.

Byte Number	0	1	2	3	Block
UID / Internal	Internal0	Internal1	Internal2	Internal3	0
Serial Number	Internal4	Internal5	Internal6	Internal7	1
Internal / Lock	Internal8	Internal9	Lock0	Lock1	2
CC	CC0	CC1	CC2	CC3	3
Data	Data0	Data1	Data2	Data3	4
Data	Data4	Data5	Data6	Data7	5
Data	Data8	Data9	Data10	Data11	6
Data	..	..	..	..	.
Data	..	..	..	..	.
Data	..	..	..	..	.
Data	..	..	..	..	.
Data	..	..	..	..	n
Lock / Reserved	..	..	..	..	.
Lock / Reserved	..		..	..	.
Lock / Reserved	..	..	..	..	k

Figure 21 - Dynamic Memory Structure [28]

The dynamic memory structure is sectioned into partitions. These partitions are internal bytes, static lock bytes, capability container, data area and reserved/dynamic lock bytes. Internal bytes are used for manufacturer information like UID and serial number. Static lock bytes are used for defining if the tag is a read/write tag or a read only tag. Changing these to the read only configuration is irreversible. The capability container is used for NDEF type 2 tag management. The data area is free for general data. For full details of the type 2 NFC tag memory structure see here [28].

Finally, the main function itself.

```
int main(void)
{
	setupNfcEmulator(tagStorage, sizeof(tagStorage));

	while(1)
	{
		checkForNfcReader();
	}
}
```

Here the NFC tag is set up using the "setupNfcEmulator" function with parameters of its content and its size and then the code is sent on an endless loop checking for an NFC reader using the "checkForNfcReader" function. These two functions will now be discussed in detail.

7.1.3 Setup NFC Emulator

This function is used only once in the code at the very start to set up the NFC tag with all its attributes. First, the microcontroller ports are initialized and internal clock bits are set.

- A clock divider is set up to deal with using different external clocks but since we are using a 13.56 crystal the divider is 1.
- The OCOA port in the ATtiny84 is set up to vary a resistance that will draw varying degrees of current from the antenna and therefore the phone. This is how the tag communicate back to the phone. The phone can detect this varying degree of current being drawn and can understand it as binary. The OCOA port varies this resistance on a subcarrier frequency of 13.56MHz/16 or 847.5kHz.
- Clear Timer on Compare Mode or CTC Mode is set up for the clock were using. CTC Mode defines when the clock resets.

- The ADC (Analogue to Digital Converter) inside the chip is enabled
- The analogue comparator is set up to compare the AIN0 port to the AIN1 port and an interrupt is flagged on a rising edge. These are the message and comparator inputs discussed in the signal processing section.

Next there is some redundancy work done. Within the NFC protocol the HALT command will always be sent with a cyclic redundancy check (CRC). This is to ensure the message is sent without any errors. Redundancy is always important when dealing with wireless communication because the signals can often be distorted as they float through the air. The "addcrc" function adds a CRC redundancy to the microcontrollers understanding of the message.

Finally, a global pointer is pointed at the start of the tag data and a global variable is given the number of bytes left in storage. These will be used later for any work to do with the tag data.

7.1.4 Check for NFC Reader

This is the main part of the code and where all the functionality to do with the NFC protocol communication happens. First, a visual representation of the handshake between the 2 devices will be presented. Then it will be discussed how this relates to the code.

The following visual shows the different states the tag goes through as it initializes communication with the phone.

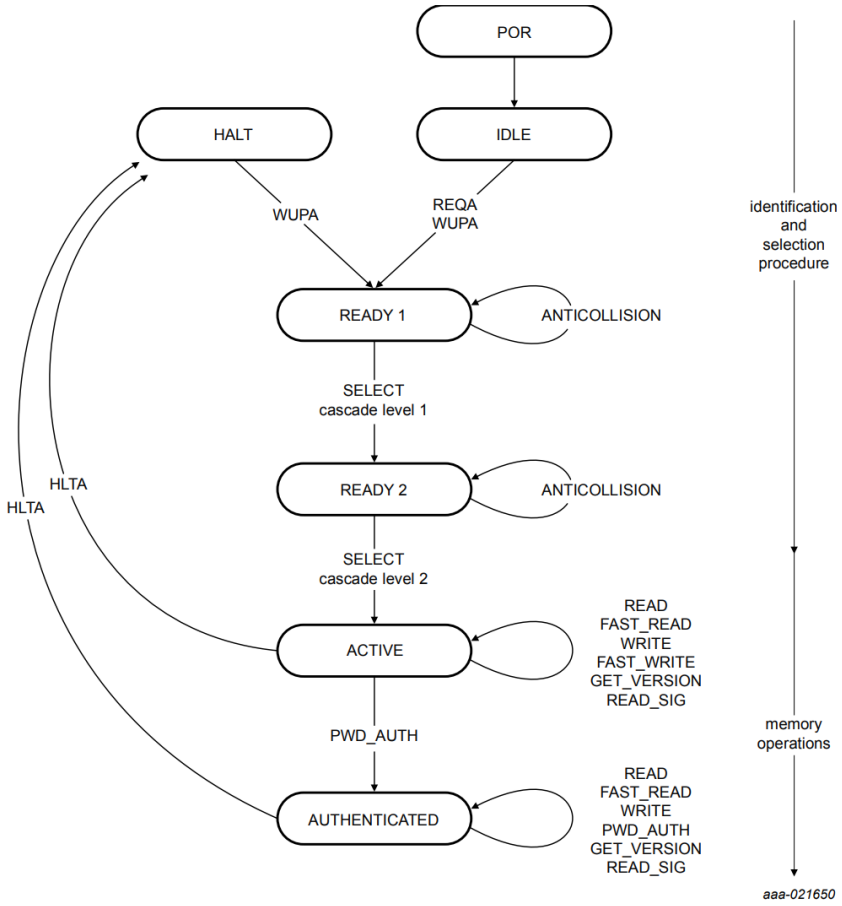


Figure 22 - Phone and NFC Tag Communication 1 [29]

The second visual gives a better idea of the conversation happening.

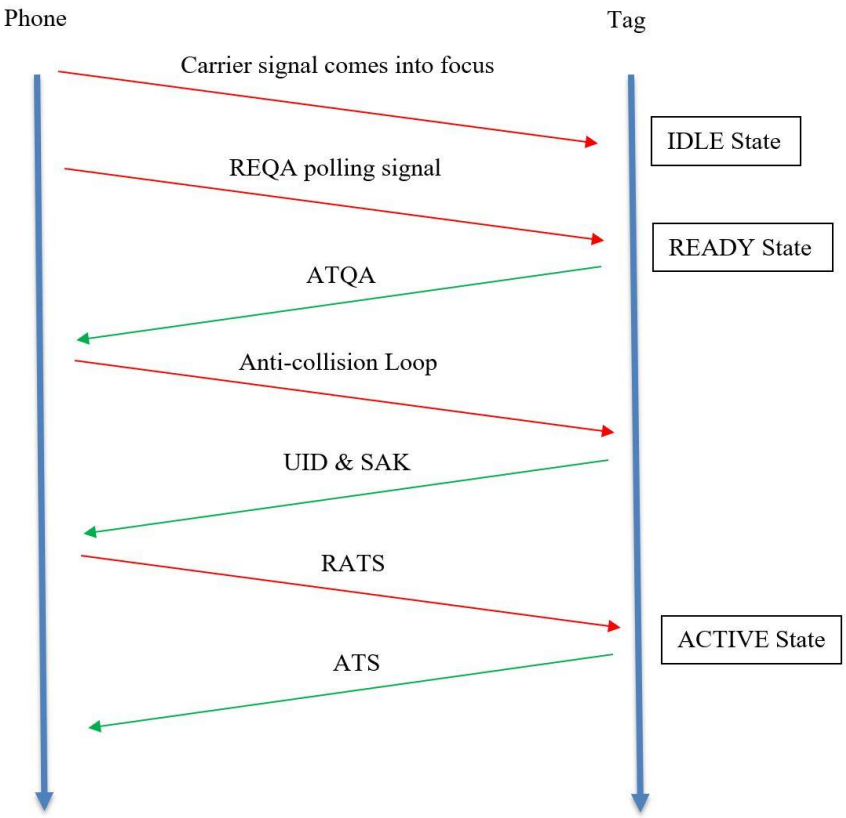


Figure 23 - Phone and NFC Tag Communication 2

It will now be discussed how the code in the function “checkForNfcReader” relates to these visuals.

- First the function looks for a carrier, if one is available it proceeds
- The tag now starts a countdown from 8 to go through the different stages of the handshake. Every time the tag sends another part of the handshake it decrements the counter and waits for a response.
- Then the input ports are deactivated to increase sensitivity and the rxMiller function is called to read in the Miller encoded signal. If that signal is a REQA polling signal (a 26 in hexadecimal as defined at the top of the nfcemulator.c code) the tag sends back a ATQA signal (value to do with manufacturer, used to avoid collisions) using its txManchester function and moves into READY state.
- An anti-collision loop will now have started, the phone will send the tag info about itself and the tag sends a UID (Unique ID) and a SAK (Select acknowledgement)
- If the tag is selected the phone sends the tag a RATS (Request Answer To Select) command and the responds with a ATS (Answer To Select) command. The tag now moves to an ACTIVE state.

- The phone will now keep the signal alive for the tag to communicate over and it will not communicate with any other tags until this communication is broken down.
- While the tag is in its ACTIVE state it uses the “SendData” function to buffer data across to the phone.
- If at any point the conversation breaks down or the tag is not selected during the anti-collision stage the function goes back to IDLE state and waits for a wake-up message.

7.2 Changes Made

This section will outline the changes I made to the code to get the NFC tag communicating with the phone. This open source code was designed to work as an NFC tag without any modifications, but this exact code didn’t work in this project without some modifications. This was mostly due to the difference in compiler and programmer used. Most of the changes made to the code in general in this project were done to implementing the sensor. All this code will be discussed in [Chapter 8 – Sensor Implementation].

To get the code compiled to use a 13.56MHz clock the AVRdude compiler was used as stated earlier. Using AVRdude meant everything had to be compiled, converted to hex files and ten uploaded onto the device. This meant it was difficult to include standard libraries. Instead the AVR library was simply put in the same folder as the main code and included using inverted commas “ ” rather than arrows < >.

```
#include "avr/io.h"
#include "nfcemulator.h"
#include "util/delay.h"
#include "nfcemulator.c"
```

Whatever way the code had been compiled before the order of the functions were not important. This meant the setupNfcEmulator function could be placed before the “addcrc” function that it uses. When the code was compiled on AVRdude it gave errors saying functions like “addcrc” were undeclared if there were not defined before their use in the code. To get the code to compile on AVRdude the order of all the functions had to be changed so that all functions were declared and defined before they were used.

The open source got this project is based on gives an option to use either 22MHz clock or a 13.56MHz clock. Using a 13.56MHz clock in this project simply meant uncommenting that option in the “nfcemulator.h” file.

```
#define F_CPU 13560000UL
```

7.3 Troubleshooting

As explained earlier there was no serial available from the microcontroller to Debug this code so it was mostly LEDs used. However, lighting LEDs during the communication between the phone and tag slowed the tag down too much and often broke the communication. One of the troubleshooting steps I took was to change the “checkForNfcReader” function to return an int rather than just void. In this way I could light different LEDs depending on the return status of the “checkForNfcReader” function. A variable called result could then be changed to different numbers as the function got to different stages of the protocol.

Chapter 8 – Sensor Implementation

Now that we have a working NFC tag communicating with the phone the next step is to add a sensor to the NFC tag. This means connecting the sensor up to the microcontroller on the bread board, configuring the microcontroller for an analogue read from the pin attached to the sensor and writing the extra software in the NFC tag code to take in sensor tag when a phone connects and send this data to the phone. There were two different types of sensors used in this project, a temperature sensor and an air-pressure sensor. The implementation will first be described in relation to the temperature sensor and then any differences in implementation will be described for the air-pressure sensor.

8.1 Sensor Chosen

There were two important factors to consider when choosing a sensor. First, it had to be very low power consumption as the unique selling point of this passive NFC sensor implementation is that it has the potential of having an extremely low power consumption and therefore lasting very long without a battery change. Second, the sensor had to be accurate enough to make it worth the hassle of using a passive NFC sensor. The ATtiny84 itself has a very relatively inaccurate temperature sensor onboard and there are already many temperature sensor hardware implementations for smartphones. If this sensor is to be worth using it would have to have an accuracy edge over other applications. For these reasons the LMT86LP temperature sensor from Texas Instruments was chosen for this project [30]. As explained on the data sheet the LMT84 and LMT85 have slightly lower power consumption and the LMT87 has slightly more accuracy but it was felt that the LMT86 had the right balance of both for this application.

8.2 Circuitry

The circuitry involved in connecting the 3-pin temperature sensor to the microcontroller was very simple. The LMT86LP temperature sensor has a VDD pin, an analogue out pin and a ground pin. The VDD pin was attached to VCC, the ground pin was attached to ground and the analogue out pin was attached to PORTA3 which was being used as the analogue input pin in this project.

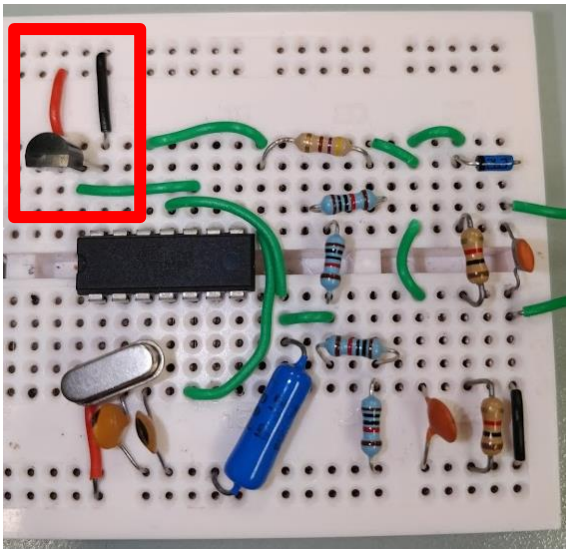


Figure 24 - Sensor Circuitry

8.3 Initializing the ADC

Considering we were programming the ATtiny84 through AVRdude and not the Arduino IDE and all our code was using the AVR library so far it was thought that it would make more sense to write the code for the sensor integration using the AVR library as well rather than the Arduino one. Especially since we had already changed a lot of low level settings in the microcontroller and it was unsure how the Arduino library would work with those changes. Also, it was required to set a reference voltage for the temperature sensor using the AVR library anyway, which is not as easy through the Arduino library. However, AVR library code is not as simple as the Arduino library. It is much lower-level and much more involved with the hardware. The simple “AnalogRead” function you call in the Arduino library does a lot more than meets the eye. These steps must be done manually with the AVR library.

The analogue to digital converter in the ATtiny84 must be set up for taking in a value from the sensor. This is done in the “ADCinit” function in the code. This function will now be explained.

The first thing that is set up is the reference voltage. The temperature sensor works by outputting a voltage through the analogue out pin that is between ground and its reference voltage. The microcontroller can then calculate the temperature by relating this analogue voltage as a percentage of the reference voltage and putting that through a simple equation given in the temperature sensors data sheet. It is very important that this reference voltage is

consistent for the temperature sensor data to be accurate. If the VCC of the microcontroller is very consistent, as in from a USB port or plug, then VCC can be used as the reference voltage. However, if the microcontroller is getting its VCC from a battery it can be very unreliable. For this purpose, there is an internal voltage reference of 1.1V set within the ATtiny84 that is very consistent and you can reference. Unfortunately, time ran out in this project before the reference voltage could be changed from VCC to this internal voltage so in this project it is VCC used. To set this reference voltage to VCC the REFS1 and REFS0 bits must be set as follows:

```
ADMUX =
       ADMUX = (0 << REFS1) |
       ADMUX = (0 << REFS0) |
```

The next thing is to set which pin is being used for the analogue input. This is done by putting the binary number for the ADC value of the pin in the 5 MUX bits in ADMUX. In this project physical pin 10 was used which is ADC pin 3 so the first 2 bits were set to 1.

```
(0 << MUX5)  |
(0 << MUX4)  |
(0 << MUX3)  |
(0 << MUX2)  |
(1 << MUX1)  |
(1 << MUX0);
```

Next the ADC must be enabled by setting the ADEN (ADC Enable) bit and the prescaler must be set. The ADC converts the analogue signal into a digital signal periodically. The frequency of this conversion depends on the clock speed. As defined on the data sheet for the ATtiny84 this frequency of conversion should be between 50KHz and 200KHz, but the clock frequency is much faster than this. The prescaler is the divider for the clock and has 7 options available i.e. 2, 4, 8, 16, 32, 64 and 128. You use a number that will divide your clock speed to something between 50KHz and 200KHz. In this project the clock speed is 13.56MHz so it must be divided by 128 to get 105.9KHz. The ADPS (ADC Precaler) bits in the ADCSRA register are set to a binary 7 to indicate 128.

```
ADCSRA =
       (1 << ADEN)  |
       (1 << ADPS2) |
       (1 << ADPS1) |
       (1 << ADPS0);
```

The final step to initializing the ADC is to define the resolution of the result values. The ATtiny84 has a 10bit ADC. An analogue to digital converter will never be perfectly accurate. It can only map an analogue measurement to its closest digital value out of a certain number of digital values. A 10bit ADC means the ATtiny84 can map an analogue input to 2^10 values therefore a number between 0 and 1023. However, it is awkward to use the extra 2 bits after the 8 bits because they are in a different register and must be put back together. For this project, because accuracy is not a priority, only 8bits of the ADC will be used, therefore given a value between 0 and 255. For example, if the reference voltage is 5V and if the temperature sensor sent back an analogue read of 2V the ADC would convert this to (2 / 5) * 255 = 102. To use only an 8bit ADC you must set the ADLAR (ADC Left Shift Adjust) bit to 1 to left shift the result. If you wanted a 10bit ADC you would set this to 0 and the result would be right shifted.

```
ADCSRB = (1 << ADLAR);
```

Full details on all the registers involved can be found on the ATtiny84 data sheet but I also found this document on this website very good for explaining how they work [31]. For more general knowledge on how ADCs in microcontrollers work I found this book very useful [32].

8.4 Functional Sensor Code

8.4.1 Code Logic

The basic logic of this code is to make a full NFC connection with the phone, take a measurement from the sensor and write this data to the tag storage where a usual NFC tag would write some text to be displayed on the phone. However, there is a slight problem with this. The device cannot take a sensor measurement before the NFC handshake has been made because in a battery-less implementation, which this project aims to be leading towards, the device would have no power until this handshake has started. As soon as the device comes in range with the phone and the NFC handshake starts the phone provides a constant signal that can power the device. Before this handshake the phone does not provide a constant signal, only a periodically pulsing polling signal and the device would not be able to take a sensor measurement. The sensor measurement also cannot be done during the handshake as it takes too long for the microcontroller to get the sensor data, do a digital conversion and write this data to the tag storage before the phone has already read the storage. If the sensor

measurement is done after the NFC handshake the tag storage has already been read and displayed by the phone. The problem is that the phone and the protocol are expecting a simple tag with prewritten storage to act on but in this project, a sensor has been added to that tag and we want to change some of that storage before the phone acts on it.

To combat this problem the code was designed to connect to the phone periodically every second holding the phones powering signal between connections. The original data stored on the device tells the phone to simply display some text, the same way a usual NFC tag does. The text stored originally is “Temperature loading…”. Then between the first connection and the second connection, with the powering signal from the phone still provided, the microcontroller takes a sensor measurement, converts it to digital, then degrees, then ASCII text and changes the text in storage to be displayed on the phone screen from “Temperature loading…” to “Temperature: XX°C”, where XX is the temperature measurement. The phone then connects a second time, exactly like the first connection, except it has different text to display. This implementation does work on any smart phone, it will just look slightly different.

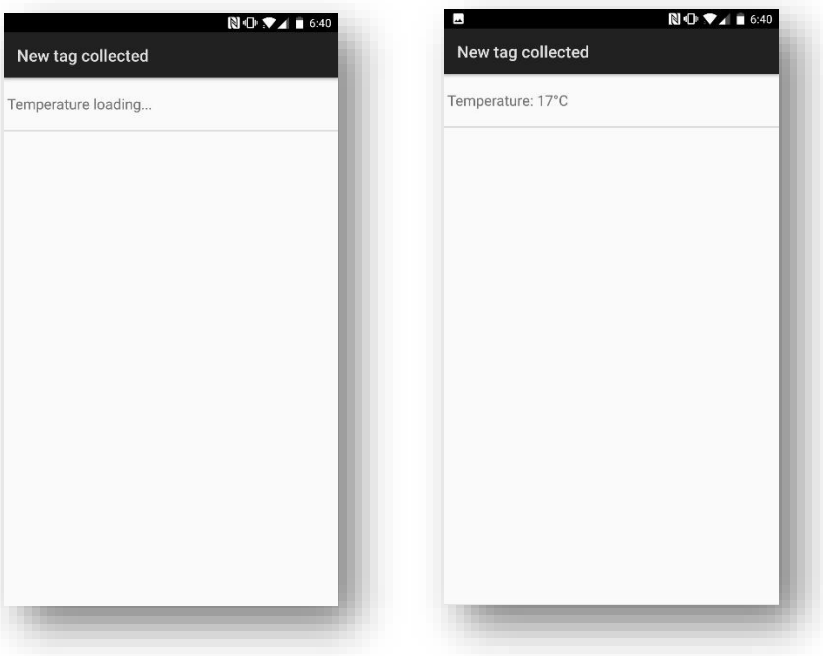


Figure 25 - Phone Display

8.4.2 Code Explained

All the functional sensor code was written in the main function in the main.c file. First, the ADC was initialized using the "ADCinit" function that was discussed above. This is just after the NFC tag has been set up and just before the while loop where the tag repeatedly looks for a reader.

The code is structured so that the measurement is only taken once the device is activated by the phone and finishes the NFC handshake. For this reason, I have made the "checkForNfcReader" function return an integer when it has finished the handshake with the phone and is now providing the circuit with a constant signal.

```
if(checkForNfcReader() == 1){
```

As soon as the NFC handshake is done the ADSC bit in the ADCSRA register is set which starts a measurement. Then a while loop is executed to wait until the ADC has done the conversion.

```
ADCSRA |= (1 << ADSC);
while (ADCSRA & (1 << ADSC) );
```

The measurement is then stored in the ADCH variable, so the value from this variable is stored in a variable called analogue_value. Then the analogue_value variable is translated into degrees Celsius using an equation. As we set the reference voltage to VCC and our battery is 3V we multiply by 3000mV and divide by 255 because were using an 8bit ADC like we did in the earlier example. The next part of the equation is given in the temperature sensors data sheet [30].

```
long int analogue_value = ADCH;
int sensor_data = ((((analogue_value*3000)/255)*40)-84000)/(-432);
```

The data being stored on the device has to input in ASCII hex so the next while loop is a simple piece of code to separate the 2 digits of the sensor_data variable so they can be input to the storage as 2 separate hex digits in position to be seen beside each other on the phone screen. This does not account for 3 digits temperatures or negative temperatures but the aim for this project was more to do with the concept device rather than a very robust or accurate temperature sensor.

```
int temp_digits[2];
int i = 1;
while (i >= 0){
	int digit = sensor_data % 10;
	sensor_data /= 10;
	temp_digits[i] = digit;
	i--;
}
```

The following code is simply about putting more ASCII hex values in the storage to turn up in place on the phone screen after the second connection to display the right message. It can be seen how we input our sensor data variables here as well.

```
tagStorage[41] = 0x3a;    //:
tagStorage[42] = 0x20;    //space

tagStorage[43] = temp_digits[0] + 0x30;
tagStorage[44] = temp_digits[1] + 0x30;

tagStorage[45] = 0xc2;    //*
tagStorage[46] = 0xb0;    //*
tagStorage[47] = 0x43;    //C
tagStorage[48] = 0x20;    //space
tagStorage[49] = 0x20;    //space
tagStorage[50] = 0x20;    //space
tagStorage[51] = 0x20;    //space
```

Chapter 9 – App Development

The app developed for this project was always going to be very time dependent. Unfortunately, as it turned out there was very little time to spare and only the bones of an NFC smart phone application were reached. This was taken from an open source project that can be found here [33].

It can recognise an NFC tag when approached and display any information on the tag. It can also display extra information about the tag like the tag ID and the length of the data on the tag. There was a learning curve to getting Android Studio up and running and understanding this code enough to debug it and get it running on a phone with the passive NFC sensor but none of the Android code was written for this project. While there was not enough time in this it was believed this was worth mentioning as it is a great inspiration for future research. A tailored interface for the passive NFC sensor would open it many new applications.

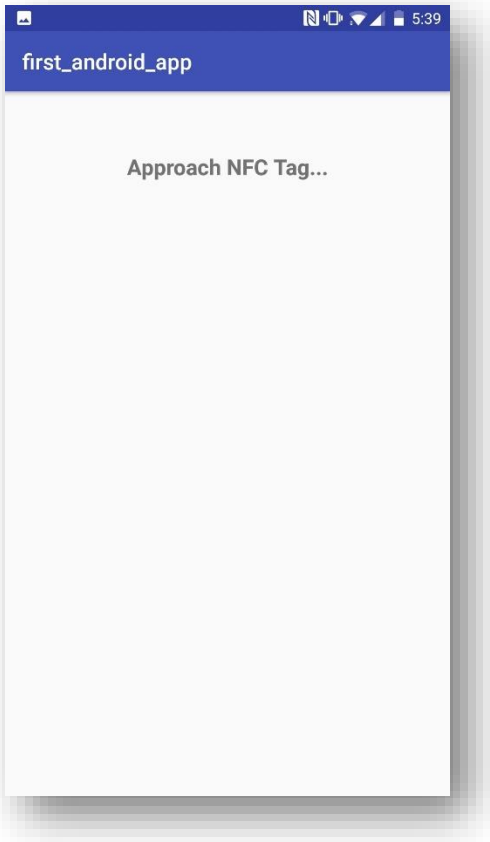


Figure 26 – NFC Android Application

Chapter 10 – Testing

10.1 Antenna Performance

10.1.1 Test Set

There are a few parameters to play with when designing an antenna coil for an application like this. The most important of these include shape, size, capacitance and number of loops. The test set was made for the following tests with these parameters in mind. To decide what value the parameters should be to optimize performance, the effect each parameter has on the performance needs to be known.

There were 6 different antennas were built to test these performance characteristics. All antennas were designed to exploit certain parameters while still resonating at 13.56MHz to be useful in the NFC protocol. For the antennas to have the same inductance so they resonate at 13.56MHz the bigger they were the less loops they had to have. For more details on the equations behind how the sizes and capacitances were chosen, see [4.2 Antenna Design].

- 3 circular antenna coils, one large with 3 loops, one medium with 4 loops and one small with 5 loops (with diameters 84mm, 52mm and 36mm respectively)
- 3 rectangular antenna coils, one large with 3 loops, one medium with 4 loops and one small with 5 loops (with width and height 80x60mm, 50x40mm and 35x25mm respectively)

All antenna coils were attached to a 56pF capacitor to resonate at 13.56MHz. The circle and rectangle antenna diameter and lengths were chosen so as the amount of wire used in the making of the antennas was the same. This way it can be said that the amount of wire used did not influence the comparison. This set is visually represented below.

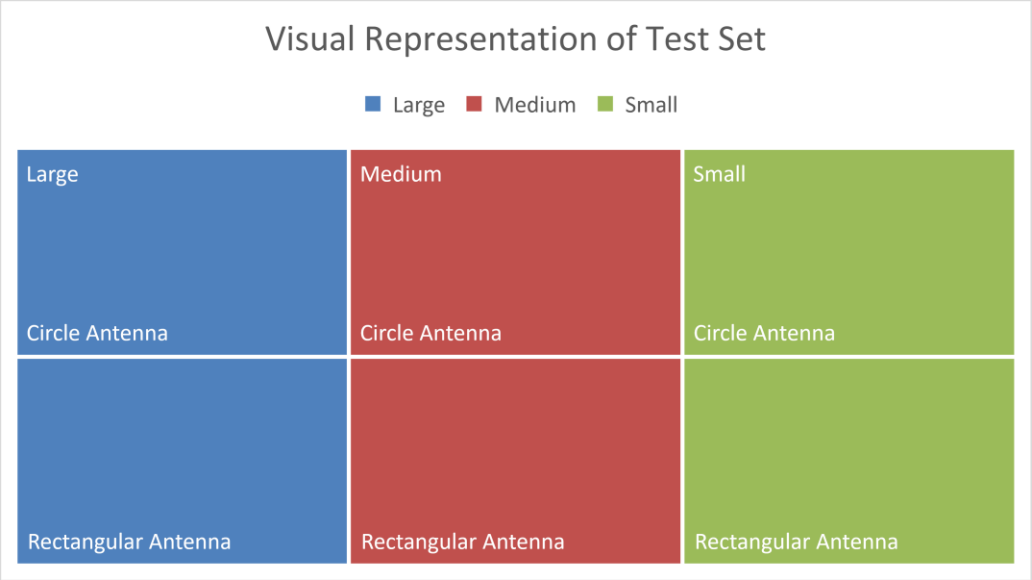


Figure 27 - Visual Representation of Test Set

With the top line representing large capacitance and the bottom line representing small capacitance. Below, a sample of the set is pictured.

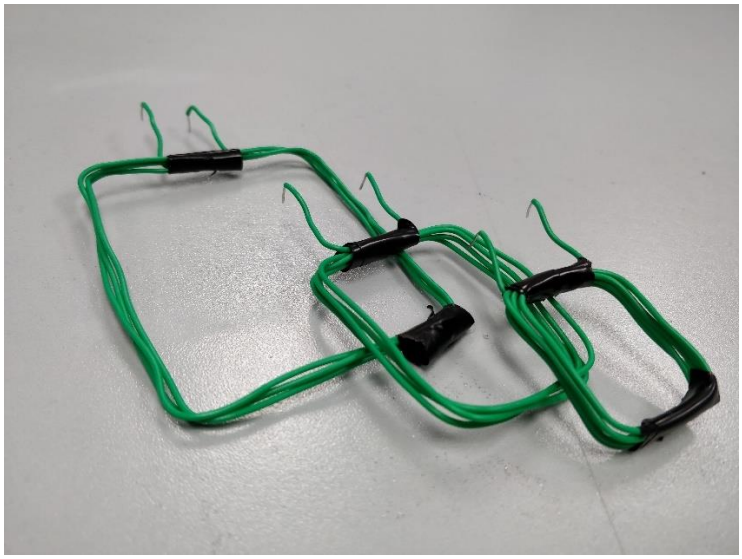


Figure 28 - A ample of the Test Set

10.1.2 Connection Performance

Connection performance in this section is understood as the ease and reliability of the antenna making a connection with the phone. The important attributes to test here are the maximum distance at which the antenna can make a connection, the maximum angle at which the antenna can make a connection and the time delay it takes for the antenna to make a connection. In this section the tests that were done to evaluate these attributes of the antenna will be explained.

The maximum distance at which the antenna could make a connection was measured by laying the antenna out flat on a table with a ruler standing straight up beside it and lowering the phone over the antenna until it made a connection and a measurement was taken. 4 tests were done for each antenna with 4 different orientations of the phone consistent across antennas and an average of the 4 measurements was taken. While the orientation of the phone was different across the horizontal plane, the phone was always kept parallel to the antenna to avoid the angle of the connection having an effect. 2 phones were used to avoid the shape of the phone antenna having an effect, 2 measurements per phone per antenna so 4 measurements per antenna. A Samsung Galaxy 7 and a One Plus 5 were the phones used.

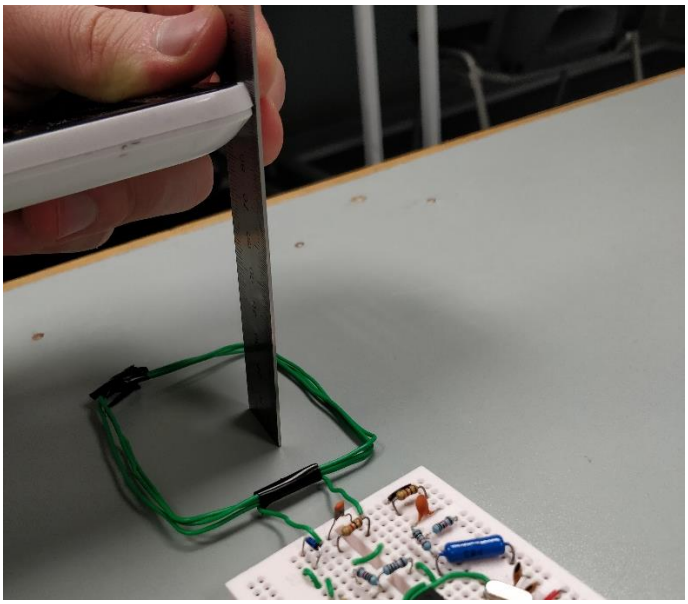


Figure 29 - Maximum Distance Test

The maximum angle at which the antenna could make a connection was measured in a similar way to the maximum distance except this time the phone was held at a consistent height perpendicularly over the antenna. Then the phone was turned closer to parallel until a connection was and a measurement of the angle was recorded using a protractor. Again, 2 phones were used taking 4 measurements per antenna at 4 different orientation in relation to the horizontal plane but consistent across antennas. Averages then calculated.

To measure the time delay for an antenna to make a connection the antenna coil was disconnected from the capacitor and the rest of the circuit and then reconnected with a phone a consistent distance above the antenna. As soon as the antenna coil was reconnected a

stopwatch was started and the amount of time it took before the phone made a connection was recorded. Again, 2 phones used with 4 measurements per antenna. Averages then calculated.

10.1.3 Power Harnessing Ability

Power harnessing ability is understood in this section as the antennas ability to harness power from the phones antenna through inductive coupling and transfer this power to the circuit for use.

To avoid the powered circuit having any effect on the current or voltage coming in from the phone the antenna coil and capacitor were separated from the circuit for this experiment. When the phone is placed over the disconnected antenna there are pulses of power coming through to the antenna as the phone sends out periodic polling signals, but it is difficult to measure accurately without the phone making a connection and providing consistent power. For this experiment the disconnected antenna was placed over an NFC tag. The phone was placed over the antenna and the NFC tag until a connection was made with the NFC tag. A Schottky diode was placed in series with the antenna to rectify the signal and a voltage measurement was taken between the ground of the antenna and the end of the diode. This voltage measurement would be an indication of the power the antenna could draw from the existing induction coupling between the phone and the tag.

To avoid distance or angle having an effect the phone was moved around at measurement until the maximum voltage was harvested through the antenna and this was recorded as the data sample. Also, for this reason it wasn't as important to use different phones for this experiment. It didn't matter where the antenna was on the phone or how different phones connected, only a comparison between antennas was desired here. What was very important however was the use of different NFC tags. A smaller antenna might harvest more energy from an existing connection between a smaller tag and the phone compared to a bigger antenna and we don't want the tag influencing this experiment. For this reason, 3 different shaped and different sized tags were used for the induction coupling.

Figure 30 - NFC Tags of Different Shape and Size

One measurement was taken per tag per antenna so three measurements per antenna. An average was then taken.

Chapter 11 – Results and Discussion

11.1 Antenna Performance

11.1.1 Connection Performance

The results for the maximum distance test were as follows:

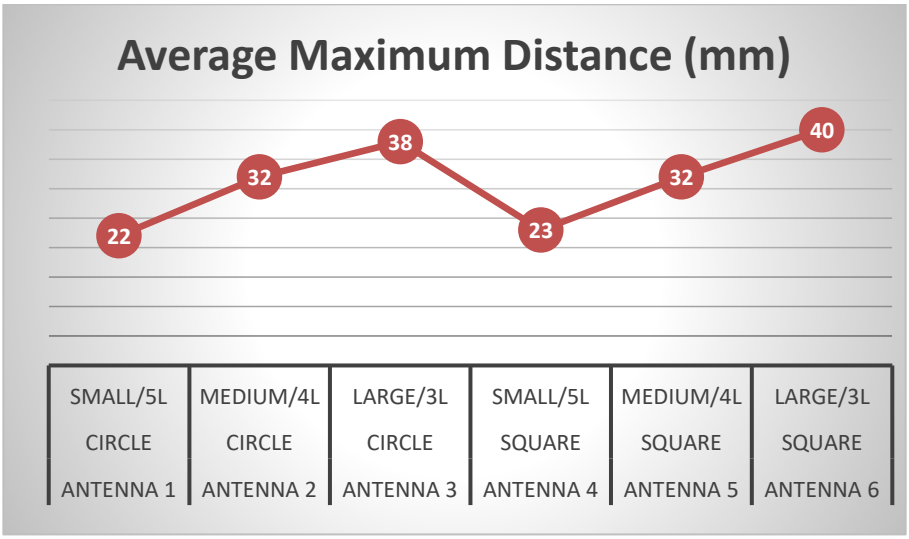


Figure 31 - Graph of Maximum Distance Test Results

The most important thing to take from this chart is that the maximum distance at which the antenna can still make a connection with the phone is very dependent on the size of the antenna. Presumably, the bigger the antenna the bigger the induction coupling loop and therefore the further away the connection can be made. It can also be noted that the rectangular antennas had a slightly higher maximum distance than the circular antenna but not enough to be sure of a causation. It was very interesting however that the circle and rectangular antennas were so close in scoring. This would explain why industry hasn't particularly favoured one over the other. For example, out of the 2 phones used to test the device one had a rectangular shaped NFC antenna and the other had a circular one. These results show that this doesn't seem to make a big different distance wise at least.

The results for the Maximum Angle test were as follows:

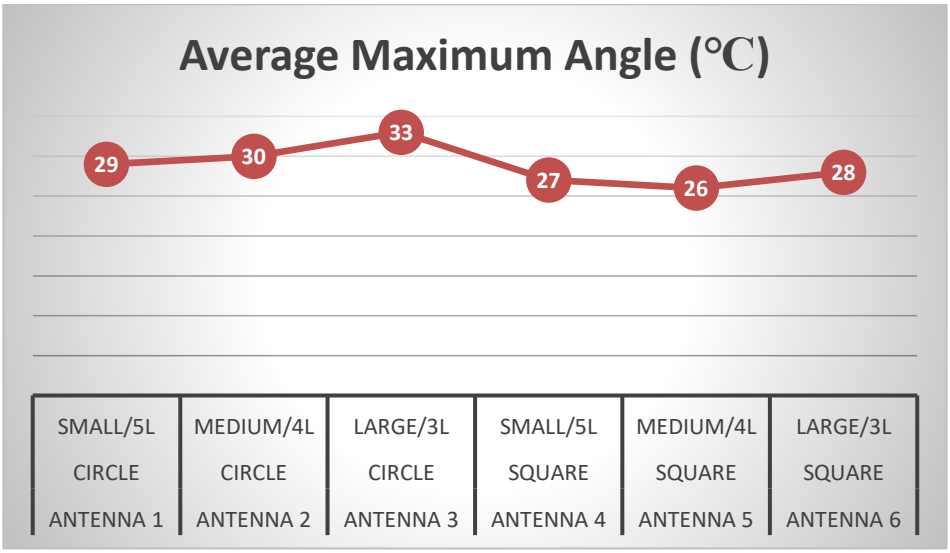


Figure 32 – Graph of Maximum Angle Test Results

According to these results there is a slight increase in the maximum angle at which the antenna can still make a connection in the circle antennas, but this is not strictly so with the rectangular antennas. The increase is not quite strong enough to make assumptions. One of the main things to take from this result is that a more accurate test would probably need to be done to confirm trends. More data samples and better equipment. However, it can be noted that the circular antennas performed consistently better than the rectangular antennas. In reflection this is probably the result of the constant changing in orientation of the phone. The circular antennas scored consistently at every orientation of the phone but the score of the rectangular antennas were more sensitive to this. The better the 2 antennas are lined up in an inductive coupling the better connection they will make so as regards the angle it probably depends on the application. A circular antenna might be better when the connection being made will always be at different orientations.

The results for the timing test were as follows:

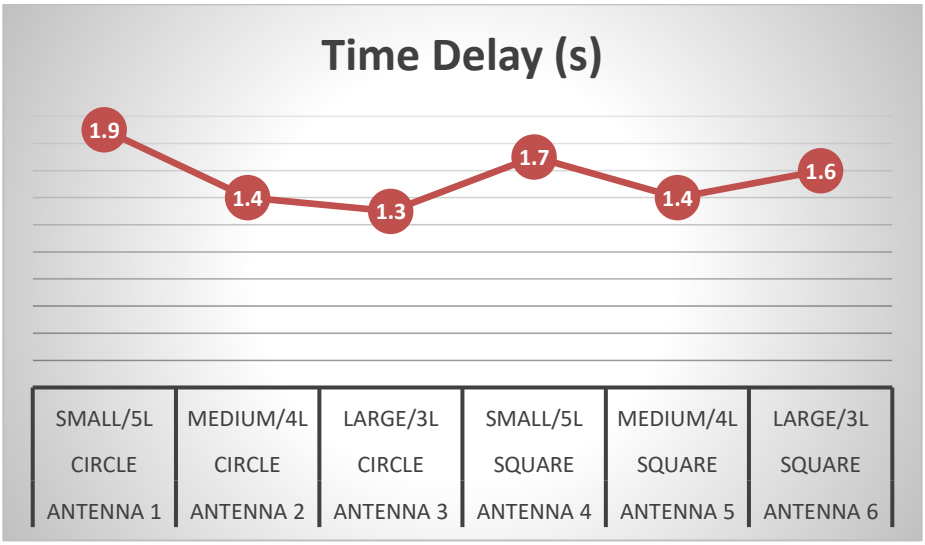


Figure 33 – Graph of Time Delay Test Results

The timing test like the angle test did not provide any concrete trends that might swing our decision. Maybe more accurate equipment or more data samples were needed. Maybe the antenna doesn't have a big effect on the time it takes the device to connect. Either way with too little between the averages and no concrete trend this graph won't be used in the decision-making process.

11.1.2 Power Harnessing Ability

The results for the power harnessing test were as follows:

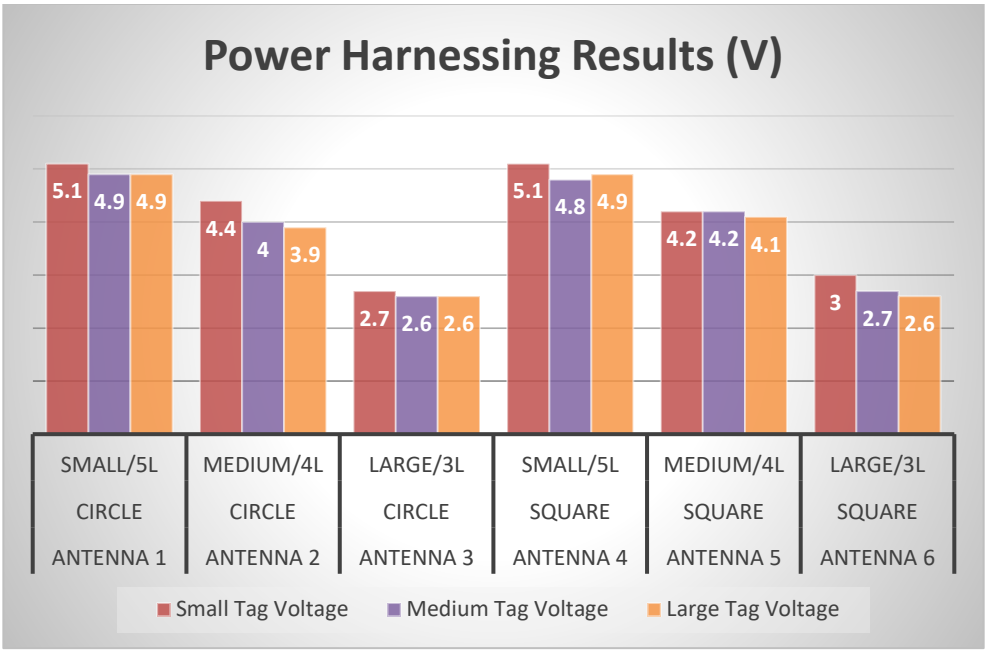


Figure 34 – Graph of Power Harnessing Test Results

The power harnessing test came up with some very useful results. Firstly, it is very noticeable that the more loops and smaller the antenna was the better it scored. Presumably, this was due to the more loops of coil and the smaller area of impact being able to concentrate more magnetic force in the one place. Secondly, it was also noticeable that the smaller the tag being used to set up the inductive loop the more voltage was harvested on every shape antenna. This is presumably for the same reasons, the smaller tags antenna setting up a stronger loop. There was one hiccup here on antenna 4 where the bigger tag with less loops did provide a stronger voltage but since this happened only once it can be ruled out to inaccuracies. Finally, it can be said again that there was very little difference between the scores of the circular antennas and the rectangular antennas. The rectangular antennas might have had a slightly higher score on average but not enough to make assumptions on.

A final useful graph to our antenna design decision-making process is to plot the most useful of our connection performance results, the maximum distance results, against the power harnessing ability results. For clarity I have sorted the antennas in order of size.

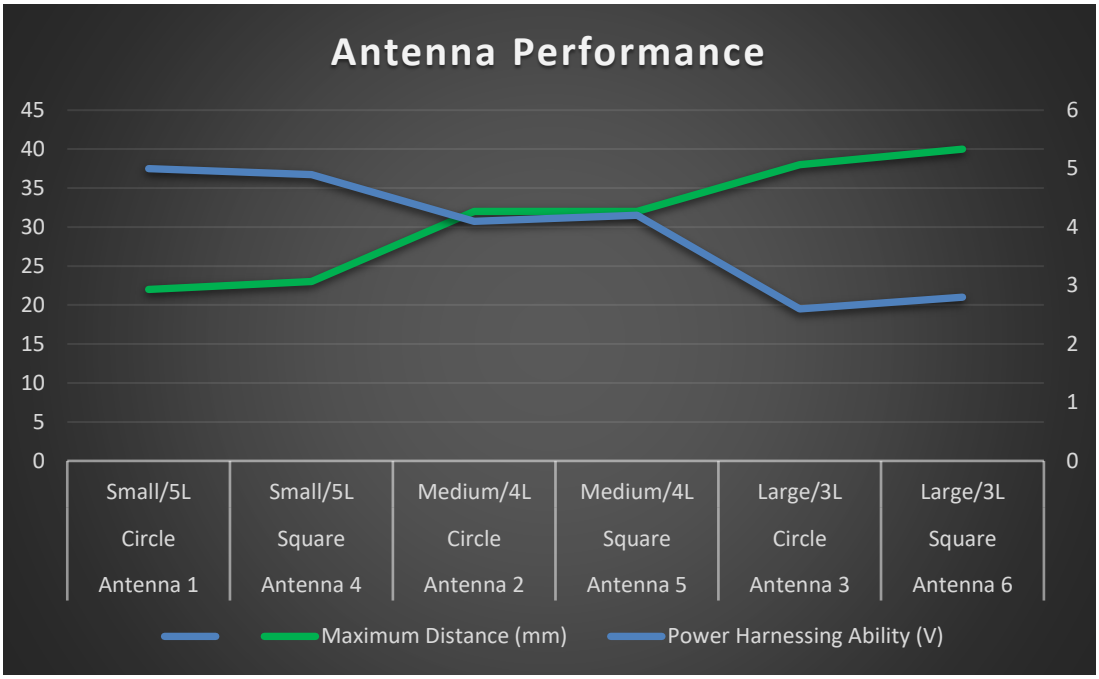


Figure 35 – Graph of Maximum Distance V Power Harnessing Ability

This graph clearly makes 2 strong points. Firstly, when considering the parameters considered here, in the scale they were considered here, a bigger antenna with less loops will provide a better maximum distance of connection but a lesser power harnessing ability. A smaller antenna with more loops will provide a better power harnessing ability but a lesser maximum connection distance. Secondly, the shape of the antenna, be it circular or rectangular has very little effect on these characteristics. Further on this graph in [Chapter 12 – Conclusion and Further Research].

Chapter 12 – Conclusion and Further Research

In this section the take away points of this project will be discussed. Specifically, a review of the device, qualities and faults and considerations for further research.

12.1 Conclusion

12.1.1 Overall Review of the Passive NFC Environmental Sensor

The aim of this project was to design a device that could easily record sensor data in areas without power supply or network connection. In that regard this project has been a success. The device built can be activated by a smart-phone through NFC, can record sensor data and transfer this data back to the phone for being displayed without network access or being plugged into the mains. Unfortunately, there was not enough time in this project to go further capability wise making the device battery-less and therefore opening it up to a host of new applications. But how well this device works on a low power battery indicates that it is a good approach to take towards creating such a device and it is a good study into how someone would go about getting this far and continue from here.

Also, some of the work done as a by-product in this assignment could well be of interest to others working in this area. In particular, the troubleshooting of an external clock on the ATtiny84, the software integration of a sensor in the NFC protocol and the performance testing of the differently shaped and sized antennas in an NFC application. These are quite basic solutions, but because NFC is such a new and fast-growing technology, there is a demand for answers to these basic questions at the moment.

12.1.2 Review of Test Results

The main learning outcomes we can take from the antenna performance test results are as follows:

For an antenna with a given capacitance and a given resonance frequency:

- A bigger antenna with less loops will be able to make connections at a greater distance but will not be able to harvest as much power from a phone
- A smaller antenna with more loops will be able to harvest more power from a phone but will not be able to make connections at a greater distance

- The shape of the antenna, be it rectangular or circular, has a relatively low effect on the antennas performance

Therefore, a bigger antenna with less loops would be recommended for a battery powered NFC application because harvesting energy isn't important, and it will have a better connection reach. For this reason, a large antenna with only 3 loops was used to build the demo addition of this device for expo day. This device is battery powered so harvesting energy is not important and it will connect quicker to people's phones during demonstrations. A rectangular shape was also chosen for this antenna since the shape is not as important and a rectangular shape suited the device aesthetically.

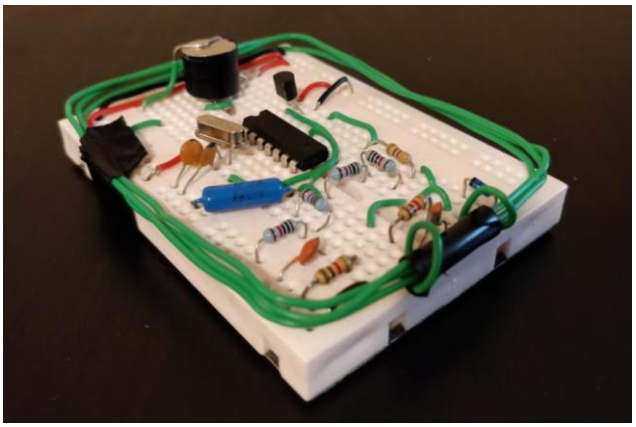


Figure 36 – Expo Demo Passive NFC Sensor

On the other hand, for further research it would be better use a smaller antenna with more loops when making a battery-less device as harvesting energy from the phone is very important.

12.2 Further Research

12.2.1 Battery-less

The most likely continued research from this project would be into making the device battery-less. This is because it would open this device up to so many more applications where a battery-less sensor is desired. Any work in this project was done with that next step in mind so completing it should be a very good foundation to build on.

The main problem with making this device battery-less would be designing a slightly more complicated circuit that could power the microcontroller with some stored energy in a

capacitor. The microcontroller does require quite a bit of current due to running at an extra high clock speed. As mentioned earlier a smaller antenna will more loops would be recommended to harness the most energy possible out of the phone.

12.2.2 Local Data Storage

Another area for further research would be the addition of local memory on the device. In certain applications it would be desirable for someone to not only get current sensor data from the device but also some history of sensor data for every time the device was activated. This would not be a complicated addition to the project. It would simply mean writing some software that would write the data recording in the EEPROM memory on the ATtiny84 every time it took a measurement. This would include software to overwrite the oldest sensor data when the memory is full. This was also an idea for this project only for there was simply not enough time.

References

[1] J. Klossner, “www.jklossner.com,” J Klossner Cartoons, 2015. [Online]. Available: http://www.jklossner.com. [Accessed 27 November 2017].

[2] D. Paret, Design Constraints for NFC Devices, John Wiley & Sons, Incorporated, 2016.

[3] L. Youbook, “Antenna Design for RFID Applications,” Microchip Technology Inc., 2003Microchip .

[4] T. Ingoe, D. Coleman and B. Jepson, Beginning NFC, Sebastopol: O'Reilly, 2014.

[5] C. Miller, “Exploring the NFC Attack Surface,” Accuvant Labs, 2012.

[6] STMicroelectronics, “ST25 NFC Guide,” STMicroelectronics, Geneva, 2016.

[7] ISO/IEC, “ISO/IEC 14443-2 - Identification cards - Part 2: Radio frequency power and signal interface,” ISO/IEC, Geneva, 1999.

[8] ISO/IEC, “ISO/IEC 14443-3 - Identification Cards - Part 3: Initialization and anticollision,” ISO/IEC, Geneva, 1999.

[9] All About Circuits, “Coil Inductance Calculator,” [Online]. Available: https://www.allaboutcircuits.com/tools/coil-inductance-calculator/. [Accessed 2 03 2018].

[10] M. Wang, Y.-X. Guo and W. Wu, “Equivalent Circuit Analysis of Inductively Coupled,” *IEEE Antennas and Propagation Society International Symposium (APSURSI),* pp. 1540-1541, 2014 .

[11] intgckts.com, “NFC Technology Aspects,” [Online]. Available: http://wireless.intgckts.com/nfc/nfc-technology-aspects/. [Accessed 03 2017].

[12] Hyper Physics, “Hyper Physics,” [Online]. Available: http://hyperphysics.phy-astr.gsu.edu/hbase/magnetic/transf.html. [Accessed March 2018].

[13] Atmel, “ATTiny84 Data Sheet,” Atmel, San Jose, 2010.

[14] IEEE, “IEEE Code of Ethics,” IEEE, 2018.

[15] B. Mammano and L. Bahra, “Safety Considerations in Power Supply Design,” Texas Instuments Incorporated, Dallas Texas, 2005.

[16] S. Pointer and J. Harrison, “Electrical Injury and Death,” Flinders University, Adelaide, 2007.

[17] HSE, “Health and Safety Statistics,” Health and Safety Executive, Caerphilly , 2007.

[18] International Electrotechnical Commission, “Safety requirements for power electronic converter systems and equipment - Part 1: General,” International Electrotechnical Commission, 2016.

[19] Matheson Gas Products, “Lower and Upper Explosive Limits for Flammable Gases and Vapors (LEL/UEL),” [Online]. Available: https://www.mathesongas.com/pdfs/products/Lower-(LEL)-&-Upper-(UEL)-Explosive-Limits-.pdf. [Accessed 03 2018].

[20] Tuv Sud, “Regulations for food packaging products and materials,” 4 November 2015. [Online]. Available: https://www.tuv-sud.com/home-com/resource-centre/publications/e-ssentials-newsletter/food-health-e-ssentials/e-ssentials-3-2015/regulations-for-food-packaging-products-and-materials. [Accessed 3 2018].

[21] Apple, “iOS Security,” Apple, Cupertino, 2018.

[22] N. Kruse, “Simple NFC,” 12 November 2013. [Online]. Available: http://blog.nonan.net/2013/11/simple-nfc.html. [Accessed 16 February 2018].

[23] B. Somanathan Nair, S. Deepa and C. Unni, A Manual of Laboratory Experiments and Workshop Practice, Kadakkal: IK International Publishing House, 2011.

[24] Jugandi, “Amplitude Modulation,” 2013. [Online]. Available: http://jugandi.com/ebooks/Wireless/03%20Amplitude%20Modulation.htm#_3.0_Amplitude_Modulation. [Accessed 2018].

[25] A. Ganesan, “create.Arduino.cc,” 21 June ` 2015. [Online]. Available: https://create.arduino.cc/projecthub/arjun/programming-attiny85-with-arduino-uno-afb829. [Accessed 30 03 2018].

[26] Engbedded, “Engbedded Atmel AVR® Fuse Calculator,” 2014. [Online]. Available: http://www.engbedded.com/fusecalc/. [Accessed 20 March 2018].

[27] Apache, “Apache Licence Version 2,” 1 January 2004. [Online]. Available: http://www.apache.org/licenses/LICENSE-2.0. [Accessed 16 February 2018].

[28] NFC Forum, “Type 2 Tag Operation Specification,” NFC Forum, Wakefield, 2011.

[29] NXP, “NTAG I2C plus: NFC Forum T2T with I2C interface, password,” NXP, Eindhoven, 2017.

[30] Texas Instruments, “LMT86 2.2-V, SC70/TO-92/TO-92S, Analog Temperature Sensors,” Texas Instruments, Dallas, 2017.

[31] maxembedded, “The ADC of the AVR,” November 2015. [Online]. Available: http://maxembedded.com/2011/06/the-adc-of-the-avr/. [Accessed 01 04 2018].

[32] J. Valvano and R. Yerraballi, Embedded Systems - Shape The World, Austin: Jonathon Valvano, 2015.

[33] Android-er, “Android NFC example, to read tag info of RFID key and card,” Februaury 2016. [Online]. Available: http://android-er.blogspot.ie/2015/10/android-nfc-example-to-read-tag-info-of.html. [Accessed 6 02 2018].

[34] N. Kruse, “blog.nonan.net,” Nonan, 12 November 2013. [Online]. Available: http://blog.nonan.net/2013/11/simple-nfc.html. [Accessed 27 November 2017].

[35] Servoflo Corporation, “www.servoflo.com,” June 2017. [Online]. Available: http://www.servoflo.com/pressure-sensors/air/high/165-absolute-digital/465-ms5541. [Accessed 27 November 2017].

[36] Android, “www.android.com,” 2017. [Online]. Available: https://developer.android.com/guide/topics/connectivity/nfc/index.html. [Accessed 27 November 2017].

[37] SecureRF Corporation, “www.securerf.com,” 2015. [Online]. Available: https://www.securerf.com/wp-content/uploads/2014/03/Spec-Sheet-LIME-Tag-NY-1.pdf. [Accessed 27 November 2017].

[38] ST Microelectronics, “www.st.com,” 2016. [Online]. Available: http://www.st.com/content/ccc/resource/technical/document/application_note/d9/29/ad/cc/04/7c/4c/1e/CD00221490.pdf/files/CD00221490.pdf/jcr:content/translations/en.CD00221490.pdf. [Accessed 27 November 2017].

[39] Texas Instruments, “www.ti.com,” June 2016. [Online]. Available: http://www.ti.com/lit/ug/tidubt8/tidubt8.pdf. [Accessed 27 November 2017].

[40] M. Barrett, K. O'Connell and A. Sung, “Analysis of transfer touch voltages in low-voltage electrical,” *Building Serivices Engineering and Technology,* vol. 1, no. 31, pp. 27-38, 2010.

[41] Sparkfun Electronics, Artist, *Attiny84.* [Art]. Sparkfun Electronics, 2018.

Appendix

1 Project Plan

Semester 1

Week 3:

- Project allocations complete work begins
- Identify early, basic, project plan for semester one including weeks to have rough milestones to do with the learning gap complete
- Search for good books/research papers/blogs to learn all around the topic of RFID and NFC and sensing in general

Week 4:

- Continue search for good books/research papers/blogs
- Start reading early books/papers found

Week 5:

- Continue reading early books/papers found
- Start looking for more specific books/research papers now with better knowledge of what's involved
- Start looking for basic RFID/NFC practical projects and circuits to put together to understand the inner workings better

Week 6:

- Finish reading early books/papers found
- Continue looking for more specific books/papers
- Reassess project plan from week 1 with better knowledge of what will be involved including practical projects deadlines, research deadlines, risk assessment work, project status report work
- First practical project: build circuit to detect NFC signal from a phone

Week 7:

- Start reading more specific books/papers
- Second practical project: Make basic android app

Week 8:

- Continue reading more specific books papers
- Third practical project: Make Android app to read and display NFC information

- Start Status Report

Week 9:

- Continue reading more specific books/papers
- Continue Status Report
- Complete Risk Assessment Form and get signed
- Fourth practical project: Design battery assisted NFC tag circuit with ATtiny (so I'm initially not dealing with power issue)

Week 10:

- Finish reading more specific books/papers
- Finish Status Report
- Fifth practical project: Design battery-less NFC tag circuit with ATtiny

Week 11:

- Catch up with anything uncomplete
- Concentrate on other modules

Week 12:

- Concentrate on other modules

Semester 2

Week 1:

- Research for and order a few simplistic and lowest power consumption sensors (probably some temperature sensors) for first sensor and circuit integration testing.
- Begin designing portable NFC power analyser with raspberry pi (for taking a quick mobile measurement of the max power received from the NFC antenna on people's phones)
- Pick out and order a few basic antennas

Week 2:

- Begin designing circuit to connect basic sensors to battery assisted NFC tag (So I'm not initially dealing with power issue.
- Complete portable NFC power analyser with raspberry pi including easily replaceable antenna slot

Week 3:

- Begin analysing people's phones for the power from there NFC port. Document results noting the differences between antenna's and phone brands
- Continue designing basic battery assisted NFC tag with sensor

Week 4:

- Continue phone NFC port power study
- Complete battery assisted NFC tag with sensor

Week 5:

- Complete phone NFC port power study including a formal report with the findings and any conclusions that can be drawn from the study
- Begin Solution 1 circuit design: A battery-less NFC tag with sensor

Week 6:

- Based on phone NFC port power study begin optimal antenna design
- Also based on study make design decisions about what power to expect from a phone NFC signal, what sensor a battery-less tag might be capable of powering
- Continue Solution 1 circuit design
- Begin Final Report

Week 7:

- If solution 1 circuit design is working
    - Begin full analysis of accuracy and efficiency and optimization
- If not begin Solution 2 circuit design with bigger or more antennas based on optimal antenna design done
- Begin Android app for reading sensor values testing with battery-assisted NFC tag
- Continue Final Report

Week 8:

- If solution 1 circuit design is working
    - Continue analysis, reporting and optimizing including design of circuit in software
    - Start using other sensors such as gas or pressure sensors
- If not continue solution 2 circuit design with bigger or more antennas
- Continue Android app for reading sensor values
- Continue Final Report

Week 9:

- If solution 1 circuit design is working
    - Complete analysis, reporting and optimization, including the printing of circuit
    - Continue and optimize design with other sensors
- If solution 2 is working

    - Continue analysis, reporting and optimization of design including designing circuit in software
    - Start using other sensors
- If neither solution is working
    - Begin Solution 3: chargeable battery included in circuit
- Continue Android app for reading sensor values
- Continue Final Report

Week 10:

- If solution 1 circuit design is working
    - Complete and optimize design with other sensors
- If solution 2 is working
    - Complete analysis, reporting and optimization of design including printing of circuit
    - Complete using other sensors
- If neither solution is working
    - Complete Solution 3: chargeable battery included in circuit
- Complete Android app for reading sensor values
- Complete Final Report

Week 11:

- Catch up with anything incomplete
- Prepare for Expo

Week 12:

- Catch up with anything incomplete
- Prepare for Expo

2 Passive NFC Sensor Code

2.1 main.c

```
/*****************************************************************************
Written and Copyright (C) by Nicolas Kruse

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*****************************************************************************/

#include "avr/io.h"
#include "nfcemulator.h"
#include "util/delay.h"
#include "nfcemulator.c"
#include <stdio.h>

uint8_t tagStorage[] = {// UID / Internal
                                        0x04, 0x96, 0xa5, 0xbF,
                                        // Serial
                                        0x89, 0xba, 0x41, 0xd5,
                                        // Internal / Lock
                                        0xa7, 0x00, 0x1b, 0x1b,
                                        // Capability Container
                                        0xe1, 0x10, 0x12, 0x00,
                                        // Data
                        0x01, 0x03, 0xa0, 0x10,
                                        0x44, 0x03, 0x1d, 0xd1,
                                        0x01, 0x19, 0x54, 0x02,
                                        0x65, 0x6e,
                                        // Text = "Temperature loading..."
                                        0x54, 0x65, 0x6d, 0x70,
                                        0x65, 0x72, 0x61, 0x74,
                                        0x75, 0x72, 0x65, 0x20,
                                        0x6c, 0x6f, 0x61, 0x64,
                                        0x69, 0x6e, 0x67, 0x2e,
                                        0x2e, 0x2e,
                                        // Lock / Reserved
                                        0xfe
};

//>>>Passive NFC Sensor Code>>>
void ADCinit()
{
   ADMUX =
            (0 << REFS1) |     // ref voltage to vcc
            (0 << REFS0) |     // ref voltage to vcc
            (0 << MUX5)  |
            (0 << MUX4)  |
            (0 << MUX3)  |
```

```
            (0 << MUX2)  |
            (1 << MUX1)  |
            (1 << MUX0);       // use ADC1 for input

  ADCSRA =
            (1 << ADEN)  |     // enable ADC
            (1 << ADPS2) |     // set prescaler to 128
            (1 << ADPS1) |
            (1 << ADPS0);

  ADCSRB =
            (1 << ADLAR);      // left shift result
}
//<<<Passive NFC Sensor Code<<<

int main(void)
{
       setupNfcEmulator(tagStorage, sizeof(tagStorage));
       ADCinit();

       while(1){
              if(checkForNfcReader() == 1){
                     //>>>Passive NFC Sensor Code>>>

                     // start ADC measurement and wait for conversion
                     ADCSRA |= (1 << ADSC);
                     while (ADCSRA & (1 << ADSC) );

                     //convert number to Celsius
                     long int analogue_value = ADCH;
                     int sensor_data = ((((analogue_value*3300)/255)*40)-84000)/(-
432);

                     //converting Celsius to ASCII hex digits
                     int temp_digits[2];
                     int i = 1;
                     while (i >= 0){
                            int digit = sensor_data % 10;
                            sensor_data /= 10;
                            temp_digits[i] = digit;
                            i--;
                     }

                     tagStorage[41] = 0x3a;    //:
                     tagStorage[42] = 0x20;    //space

                     tagStorage[43] = temp_digits[0] + 0x30;
                     tagStorage[44] = temp_digits[1] + 0x30;

                     tagStorage[45] = 0xc2;    //*
                     tagStorage[46] = 0xb0;    //*
                     tagStorage[47] = 0x43;    //C
                     tagStorage[48] = 0x20;    //space
                     tagStorage[49] = 0x20;    //space
                     tagStorage[50] = 0x20;    //space
                     tagStorage[51] = 0x20;    //space

                     //<<<Passive NFC Sensor Code<<<
              }
}    }
```

2.2 nfcemulator.c

```
/*****************************************************************************
Written and Copyright (C) by Nicolas Kruse

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*****************************************************************************/

#include <avr/io.h>
#include <util/delay.h>
#include "nfcemulator.h"

static uint8_t RX_MASK[18] = {0, 1, 0, 2, 0, 4, 0, 8, 0, 16, 0, 32, 0, 64, 0,
128, 0, 0};
static uint8_t TX_MASK[8] = {1, 2, 4, 8, 16, 32, 64, 128};
static uint16_t FDT_DELAY[2] = {FDT_DELAY_BIT0, FDT_DELAY_BIT1};

static uint8_t REQA[1] = {0x26};
static uint8_t WUPA[1] = {0x52};
static uint8_t HLTA[4] = {0x50,  0x00,  0x57,  0}; //'50' '00' CRC_A
static uint8_t ATQA[2] = {0x44, 0x00}; //Anticollision with udi size = double
static uint8_t SEL_CL1[2] =  {0x93, 0x20};
static uint8_t SEL_CL2[2] =  {0x95, 0x20};
static uint8_t CT_UID1[5] = {0x88, 0x04, 0xE3, 0xEF, 0}; //uid0 uid1 uid2 uid3
BCC
static uint8_t UID2[5] = {0xA2, 0xEF, 0x20, 0x80, 0}; //uid3 uid4 uid5 uid6 BCC
static uint8_t SAK_NC[3] = {0x04, 0xDA, 0x17}; //Select acknowledge uid not
complete
static uint8_t SAK_C[3] = {0x00, 0xFE, 0x51}; //Select acknowledge uid complete,
Type 2 (PICC not compliant to ISO/IEC 14443-4)
static uint8_t READ[1] = {0x30};
static uint8_t WRITE[1] = {0xA2};

uint8_t *sto; //Pointer to tag content
uint16_t stoSize; //Number of available bytes on the tag

uint8_t buffer[64];
uint8_t rCount = 0;

void addCrc16(uint8_t *Data, uint8_t Length)
{
	uint8_t ch;
	uint16_t wCrc = 0x6363; // ITU-V.41

	do {
		ch = *Data++;

		ch = (ch^(uint8_t)((wCrc) & 0x00FF));
		ch = (ch^(ch<<4));
```

```
            wCrc = (wCrc >> 8)^((uint16_t)ch <<
8)^((uint16_t)ch<<3)^((uint16_t)ch>>4);
      } while (--Length);

      *Data = (uint8_t) (wCrc & 0xFF);
      Data++;
      *Data = (uint8_t) ((wCrc >> 8) & 0xFF);
}

void addBcc(uint8_t *Data) //add exclusive-OR of 4 bytes
{
      Data[4] = Data[0] ^ Data[1] ^ Data[2] ^ Data[3];
}

void waitForBitend()
{
      while(!(TIFR1 & (1<<OCF1A))); //Wait till end of bit-time
      TIFR1 |= (1<<OCF1A);
}

void setupNfcEmulator(uint8_t *storage, uint16_t storageSize)
{
      //clock divider for 8 bit timer0: clk/1 -> 13.5225 MHz
      TCCR0B |= (1<<CS00);

    //8 bit timer0: Toggle OC0A on Compare Match and CTC-Mode
    //for 847.5 kHz subcarrier
    TCCR0A |= (1<<COM0A0) | (1<<WGM01);

      //set up 847.5 kHz subcarrier for sending (8 bit timer0)
      OCR0A = SUBC_OVF;

      //CTC-Mode and no clock divider for 16 bit timer1: clk/1
      TCCR1B = (1<<WGM12) | (1<<CS10);

      //Setup Analog Comparator, Enable (ACD), Set Analog Comparator
      //Interrupt Flag on Rising Output Edge (ACIS0, ACIS1)
      ACSR = (0<<ACD) | (1<<ACIS0) | (1<<ACIS1);

      addCrc16(HLTA, 4);
      addBcc(CT_UID1);
      addBcc(UID2);

      stoSize = storageSize;
      sto = storage;
}

#if (F_CPU == RFID_FREQU)
void waitForOneBitTime()
{
      waitForBitend();
}
#elif (F_CPU == 22000000UL)
//Skip every 17 bit times 1 cycle
void waitForOneBitTime()
{
      if (rCount < 7)
      {
            OCR1AL = CLC_PBIT / 2 - 1;
            rCount++;
```

```
	}
	else
	{
		OCR1AL = CLC_PBIT / 2 - 2;
		rCount = 0;
	}
	waitForBitend();
}
#elif (F_CPU == 13592500UL)
//Add every 6 bit times 1 cycle
void waitForOneBitTime()
{
	if (rCount < 6)
	{
		OCR1AL = CLC_PBIT / 2 - 1;
		rCount++;
	}
	else
	{
		OCR1AL = CLC_PBIT / 2;
		rCount = 0;
	}
	waitForBitend();
}
#else
#error "Not supported frequency, please add support if possible"
#endif

void txManchester(uint8_t *data, uint8_t length){
	uint8_t txBytePos = 0;
	uint8_t txbitPos = 0;
	uint8_t parity = 0;

	TIFR1 |= (1<<OCF1A);

	//Send SOC
	waitForBitend();
	DDRB |= (1<<2);
	OCR1A = CLC_PBIT / 2 - 1; //Set Hi- and Low-Bit
	waitForOneBitTime();
	DDRB &= ~(1<<2);

	do
	{
		if (TX_MASK[txbitPos] & data[txBytePos])
		{
			waitForOneBitTime();
			DDRB |= (1<<2);
			parity ^= 1;
			waitForOneBitTime();
			DDRB &= ~(1<<2);
		}
		else
		{
			waitForOneBitTime();
			DDRB &= ~(1<<2);
			waitForOneBitTime();
			DDRB |= (1<<2);
		}
```

```
			txbitPos++;

			if (txbitPos > 7)
			{
				if (parity)
				{
					waitForOneBitTime();
					DDRB &= ~(1<<2);
					waitForOneBitTime();
					DDRB |= (1<<2);
				}
				else
				{
					waitForOneBitTime();
					DDRB |= (1<<2);
					waitForOneBitTime();
					DDRB &= ~(1<<2);
				}

				txBytePos++;
				txbitPos=0;
				parity = 0;
			}
		}
		while(txBytePos < length);

		//Send EOC
		waitForOneBitTime();
		DDRB &= ~(1<<2);
}

inline void resetRxFlags()
{
		TCNT1 = 0;
		TIFR1 |= (1<<OCF1A); //Clear Timer Overflow Flag
		ACSR |= (1<<ACI); //Clear Analog Comparator Interrupt Flag
}

uint8_t rxMiller(){
		#if (F_CPU > 16000000)
			uint16_t t; //For hi cpu clock a 8 bit variable will overflow (CLCM
> 0xFF)
		#else
			uint8_t t; //For low cpu clock computing time is to low for 16bit a
variable
		#endif

		uint16_t cDown = 0x0FFF;
		uint8_t bytePos = 0;
		uint8_t hbitPos = 0;

		OCR1A = CLCL-1;
		buffer[0] = 0;

		//Wait for transmission end if there is data arriving
		do
		{
			if (ACSR & (1<<ACI)) resetRxFlags();
		}
		while(~TIFR1 & (1<<OCF1A));
```

```
	//Wait for transmission end if there is data arriving
	do
	{
		if (TIFR1 & (1<<OCF1A))
		{
			TCNT1 = 0;
			TIFR1 |= (1<<OCF1A);
			cDown--;
			if (!cDown) break;
		}
	}
	while(~ACSR & (1<<ACI));

	if (cDown)
	{
		resetRxFlags();
		do
		{
			if ((ACSR & (1<<ACI)) && (TCNT1 > 1))
			{
				t = TCNT1;
				resetRxFlags();

				hbitPos += (t > CLCS) + (t > CLCM);

				if(hbitPos > 17)
				{
					bytePos++;
					hbitPos -= 18;
					buffer[bytePos] = 0;
				}

				buffer[bytePos] |= RX_MASK[hbitPos];

				hbitPos += 2;
			} //34 or 41 (hbitPos > 17) click cycles
		}
		while(~TIFR1 & (1<<OCF1A));
	}

	OCR1A = FDT_DELAY[hbitPos & 1]; //Set delay for answer
	TIFR1 |= (1<<OCF1A);

	if (hbitPos > 7) bytePos++;

	return bytePos;
}

void sendData(uint8_t block){
	uint8_t i = 0;
	uint16_t pos = (uint16_t)block * 4;

	for(i=0; i < 16; i++)
	{
		if (pos >= stoSize)
		{
			buffer[i] = 0;
		}
		else
```

```
		{
			buffer[i] = sto[pos];
		}

		pos++;
	}

	addCrc16(buffer, 16);
	txManchester(buffer, 18);
}

void receiveData(uint8_t block)
{
	uint8_t i = 0;
	uint16_t pos = (uint16_t)block * 4;
	uint8_t crc1 = buffer[6];
	uint8_t crc2 = buffer[7];

	addCrc16(buffer, 6);

	if (buffer[6] == crc1 && buffer[7] == crc2)
	{
		for(i=2; i < 6; i++) //byte 2-5 contains Data
		{
			if (pos < stoSize) sto[pos] = buffer[i];
			pos++;
		}

		buffer[0] = 0x0A; //ACK
		txManchester(buffer, 1);
	}
	else
	{
		buffer[0] = 0x01; //NAK for CRC error
		txManchester(buffer, 1);
	}
}

int checkForNfcReader()
{
	uint8_t bytes = 1;
	uint8_t state = 0;
	uint8_t cdow = 8;
	uint8_t result = 0;

	if (ACSR & (1<<ACI)) //13.56 MHz carrier available?
	{
		PORTA &= ~(1<<2); //Deactivate pull up to increase sensitivity

		while(cdow > 0)
		{
			bytes = rxMiller();

			if ((state & 7) == S_READY)
			{
				if (buffer[0] == SEL_CL1[0] && buffer[1] == SEL_CL1[1]
)
				{
					txManchester(CT_UID1, sizeof(CT_UID1));
				}
```

```
                        else if (buffer[0] == SEL_CL2[0] && buffer[1] ==
SEL_CL2[1] )
                        {
                              txManchester(UID2, sizeof(UID2));
                        }
                        else if (buffer[0] == SEL_CL1[0] && buffer[1] == 0x70
)
                        {
                              txManchester(SAK_NC, sizeof(SAK_NC));
                        }
                        else if (buffer[0] == SEL_CL2[0] && buffer[1] == 0x70
)
                        {
                              txManchester(SAK_C, sizeof(SAK_C));
                              state++; //Set state to ACTIVE
                        }
                        else
                        {
                              state &= 8; //Set state to IDLE/HALT
                        }
                  }
                  else if ((state & 7) == S_ACTIVE)
                  {
                        if (buffer[0] == READ[0])
                        {
                              sendData(buffer[1]);
                        }
                        else if (buffer[0] == WRITE[0])
                        {
                              receiveData(buffer[1]);
                        }
                        else if (buffer[0] == HLTA[0] && buffer[2] == HLTA[2]
)
                        {
                              state = S_HALT;
                        }
                        else if(bytes)
                        {
                              state &= 8; //Set state to IDLE/HALT
                        }
                  }
                  else if (bytes == 1 && (buffer[0] == REQA[0] || buffer[0] ==
WUPA[0])) //state == S_IDLE
                  {
                        txManchester(ATQA, sizeof(ATQA));
                        state = (state & 8) + S_READY; //Set state to READY
                        result = 1;
                  }

                  cdow -= (bytes == 0);
            }
            PORTA |= (1<<2); //Activate pull up to prevent noise from toggling
the comparator
      }
      ACSR |= (1<<ACI); //Clear comparator interrupt flag
      return result;
}
```

2.3 nfcemulator.h

```
/*****************************************************************************
Written and Copyright (C) by Nicolas Kruse

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*****************************************************************************/

#define F_CPU 13560000UL
//#define F_CPU 22000000UL

#define RFID_FREQU 13560000UL
#define CLC_PBIT (uint16_t)(128.0 * F_CPU / RFID_FREQU + 0.5)
#define CLCS CLC_PBIT * 5 / 4
#define CLCM CLC_PBIT * 7 / 4
#define CLCL CLC_PBIT * 9 / 4

#define FDT_DELAY_BD9 (uint16_t)(9.0 * 128 * F_CPU / RFID_FREQU - 1) //Nr of
cycles-1 for 9 bit durations
#define FDT_DELAY_BIT0 (FDT_DELAY_BD9 + 20 - CLCL)
#define FDT_DELAY_BIT1 (FDT_DELAY_BD9 + 84 - CLCL)

#define SUBC_OVF (uint8_t)(F_CPU / 847500.0 / 2.0 + 0.5 - 1) //847500 Hz
Subcarrier

#define S_IDLE 0
#define S_READY 1
#define S_ACTIVE 2
#define S_HALT 8
#define S_READY_H 9
#define S_ACTIVE_H 10

void setupNfcEmulator(uint8_t *storage, uint16_t storageSize);
int checkForNfcReader();
```

3 NFC Android App Code

3.1 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.niallquirke.first_android_app">

    <uses-permission android:name="android.permission.NFC"/>
    <uses-feature android:name="android.hardware.nfc" android:required="true"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
            <action android:name="android.nfc.action.TAG_DISCOVERED" />
            <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

3.2 activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="40dp"
        android:layout_gravity="center_horizontal"
        android:text="Approach NFC Tag..."
        android:textSize="20dp"
        android:textStyle="bold"/>

    <TextView
        android:id="@+id/info"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

3.3 MainActivity.java

```
package com.niallquirke.first_android_app;

import android.content.Intent;
import android.nfc.NfcAdapter;
import android.nfc.Tag;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    private NfcAdapter nfcAdapter;
    TextView textViewInfo;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textViewInfo = (TextView)findViewById(R.id.info);

        nfcAdapter = NfcAdapter.getDefaultAdapter(this);
        if(nfcAdapter == null){
            Toast.makeText(this,
                    "NFC NOT supported on this devices!",
                    Toast.LENGTH_LONG).show();
            finish();
        }else if(!nfcAdapter.isEnabled()){
            Toast.makeText(this,
                    "NFC NOT Enabled!",
                    Toast.LENGTH_LONG).show();
            finish();
        }
    }

    @Override
    protected void onResume() {
        super.onResume();

        Intent intent = getIntent();
        String action = intent.getAction();

        if (NfcAdapter.ACTION_TAG_DISCOVERED.equals(action)) {
            Toast.makeText(this,
                    "onResume() - ACTION_TAG_DISCOVERED",
                    Toast.LENGTH_SHORT).show();

            Tag tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
            if(tag == null){
                textViewInfo.setText("tag == null");
            }else{
                String tagInfo = tag.toString() + "\n";

                tagInfo += "\nTag Id: \n";
                byte[] tagId = tag.getId();
                tagInfo += "length = " + tagId.length +"\n";
                for(int i=0; i<tagId.length; i++){
                    tagInfo += Integer.toHexString(tagId[i] & 0xFF) + " ";
```

```
                }
                tagInfo += "\n";

                String[] techList = tag.getTechList();
                tagInfo += "\nTech List\n";
                tagInfo += "length = " + techList.length +"\n";
                for(int i=0; i<techList.length; i++){
                    tagInfo += techList[i] + "\n ";
                }

                textViewInfo.setText(tagInfo);
            }
        }else{
            Toast.makeText(this,
                    "onResume() : " + action,
                    Toast.LENGTH_SHORT).show();
        }
    }
}
```