6-2017

Wireless Battery-less Sensor Device Using NFC

Matthew Mueggenberg
St. Cloud State University

Wireless Battery-less Sensor Device Using NFC

By

Matthew Mueggenberg

A Starred Paper

Submitted to the Graduate Faculty of

Saint Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Electrical Engineering

June, 2017

Acknowledgements

I would like to give my appreciation to Dr. Yi Zheng for his encouragement, help and support throughout my academic career. He has tremendous knowledge and his passion towards engineering is contagious and has sparked my interest in engineering.

I would like to thank my committee members Dr. Aiping Yao and Dr. Ling Hou for being excellent teachers and for providing guidance during my project and class work.

I would like to thank my friends and classmates Vini and Cheng for their help and conversation in the lab.

I want to express my gratitude toward my parents for their love and support and always encouraging me to strive to do my best.

Finally, I would like to thank my wife Rheyannon for all her love and support throughout my endeavors. She has provided me motivation and inspiration during my work.

Table of contents

List of Tables

List of Figures

Chapter 1: Introduction

NFC, or Near Field Communications, is a short range wireless communications standard which is a specialized subset of RFID [1]. This star paper project is broken into two parts: NFC as a research topic, and the development of a demonstrational NFC system. As a research topic, NFC technology was looked at in depth to examine the physical ideas and engineering practices used to implement designs applying NFC technology. As an exploration and demonstration of the technology a wireless and battery-less sensor device was developed.

NFC was chosen as a topic due to the intriguing nature of NFC. It is a short-range communication system by design, and can wirelessly transmit information at a relatively low data rate. Low data rates and short range are usually not desirable when selecting a wireless communication method. However, the energy transfer capability of the system allows for a battery-less device. These characteristics give NFC a specific application set which is useful for many design cases.

Research on the topic of NFC includes embedded system design for NFC, as well as more general NFC topics. The NFC standard including modulation method and encoding scheme were investigated. Common applications and uses, passive and active examples, and the future of the technology are areas of interest.

The project includes a hardware component to design, implement, and test a wireless battery-less NFC sensor device. The emphasis of the project is placed on design and implementation of this device. To communicate with the NFC sensor device, a simple supporting Android application was developed.

The goals and aims of the project are to gain in depth knowledge and insight on NFC, as well as the design process in general. Demonstration of the wireless communication and power transfer capabilities is also a primary goal. Experience was gained along many development aspects, including creating a power budget, component selection, PCB trace antenna design, schematic, and layout.

Chapter 2: NFC Background

NFC serves as an alternative wireless communication method with short range [1]. NFC is short range by design, with a maximum distance of 10cm [2]. The center frequency of the signal is 13.56MHz, and the data rate is up to 424kBit/s [1]. The modulation scheme used by NFC is amplitude shift keying, or ASK [3]. NFC is a 1 to 1 communication method, which uses a master and a slave hierarchy. The master provides the radio frequency (RF) signal, and waits for a response for the slave. In the NFC standard, the master is referred to as the initiator, and the slave is referred to as the device [3]. Reasons to use NFC include simplicity, accessibility, security, cost efficiency, versatility, and energy efficiency [1]. NFC has two communication modes, three operating modes, and four tag types.

There are two types of NFC communication modes, active mode or passive mode. In the active communication mode, the initiator and target each generate their own electromagnetic field. Both NFC devices in this mode have their own power supply. Only one side will communicate at a time to avoid collisions. This communication scheme is used with the active peer-to-peer operation mode. In passive communication, only the initiating device provides the carrier field. In this case, the target device, or tag, will derive power from the electromagnetic field. The target responds to the initiator through load modulation by adjusting the antenna impedance. [1]

There are three NFC operating modes: reader/writer, peer to peer, and card emulation [1]. In reader/writer mode, an NFC enabled device reads or writes to an NFC tag using the ISO 14443 or FeliCa standards [1]. This communication mode occurs between a reader and a tag, or a smartphone and a tag [1]. Reader/writer is typically done using passive NFC. Reader/writer mode is an open mode, and is the NFC operating mode used for the hardware portion of this project.

Peer to peer mode is also an open mode, and occurs between 2 NFC enabled devices [1]. These devices exchange data based on the ISO 18092 standard [1]. Peer to peer also may have an active or passive NFC mode [1]. A typical example of peer to peer mode is two NFC enabled smartphones exchanging data [1].

The third NFC operational mode is card emulation mode. Unlike the previous two, card emulation mode is a secure NFC mode [1]. The NFC device emulates passive mode to comply with the ISO 14443 standard [1]. A typical example of card emulation mode is a smartphone emulating a smart card at the point of sale of an NFC enabled register [1].

There are 4 tag types as defined by the NFC forum, types 1-4 [1]. Write protection on a tag prevents a reader from erasing or overwriting the tag, making it a read-only tag [1]. Type 1 fulfills the ISO 14443A standard [2]. The transfer speed is 106 kbps, with 96 bytes to 2 kilobytes of memory available [1]. These tags can be configured for read or read and write [2]. Type 2 tags are almost the same as Type 1 tags, and add the benefit of anti-collision support [2]. Type 3 tags fulfill the Japanese standard FeliCa, and have higher data rates of 212 kbps or 424 kbps [1]. Type 3 tags also support anti-collision, and are read-only or read and write configurable [2]. Finally, Type 4 tags have an expanded memory of 2, 4, or 8 kilobytes [2]. They fulfill the ISO 14443A standard, and have transfer speeds of 106 kbps, 212 kbps, or 424 kbps [1]. These tags have support for anti-collision [2].

NFC can be compared to other short range wireless data transfer technologies, such as RFID, QR, or Bluetooth. NFC is a very similar technology to RFID, and is a subset of RFID [1]. NFC has a much more limited range compared to RFID, which can communicate in greater distances using active communication [1]. As NFC is a subset based on similar principles as RFID, NFC readers can read and write to some RFID tags [2]. NFC builds on RFID by increasing the data rage and memory storage [2].

NFC has several overlapping applications when compared to QR [1]. QR stands for quick response, and it works using a camera and a two-dimensional bar code [1]. NFC does not need an application running to use, unlike QR [1]. NFC also does not need a direct line of sight which QR requires to operate [1]. In addition, NFC is much more secure when compared to QR [1]. QR has the advantages that it can be read by any smartphone with a camera, and it is cheaper to implement using a barcode [1].

NFC can be used as an initiator for faster pairing of two Bluetooth endpoints [1]. Both Bluetooth and NFC are short range wireless standards, but Bluetooth provides the advantages of having a much greater range and a much faster data rate [1]. In this way, Bluetooth is much better for sustained data transfers [1]. NFC has the advantage of using passive devices [1].

The NFC forum is the leading effort for establishing standards for NFC [1]. The NFC forum was established in 2004, and is comprised of over 180 companies [1]. These companies work together to develop specifications and to provide approval and certification of NFC devices [1]. The N mark is the notification symbol for official NFC enabled devices [1]. Other NFC contributors include the Smart Card Alliance, which was established in 2001 [1]. The SIMAlliance, established in 2000, provides guidance on the Secure Element (SE) implementation [1]. Another type of tag is the Mifare classic tag [1]. Some NFC enabled devices can read Mifare classic tags, while others cannot [2]. NFC forum official tags are in widespread use, while many devices are dropping support for the Mifare classic tags [2].

There are a wide variety of applications for NFC. Some business and marketing applications include business cards, smart posters, stickers, wrist bands, and promotional materials [1]. NFC systems can be used to unlock doors at businesses or hotels as wireless key card access [1]. Figure 2.1 shows an example of a wireless key card access system. As an example, the hotel room number and key information are stored in a smart card, either a physical card or a user smartphone [1]. When the card is in close proximity to the lock, the information is read from the smart card and verified to be the correct key, and the door is unlocked [1].

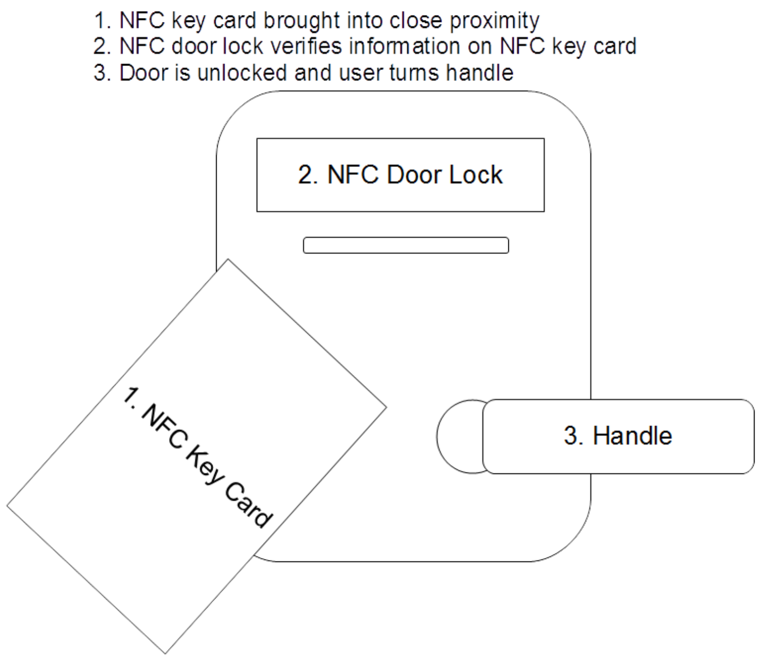

1. NFC key card brought into close proximity
2. NFC door lock verifies information on NFC key card
3. Door is unlocked and user turns handle
2. NFC Door Lock
1. NFC Key Card
3. Handle

Figure 2.1: NFC Key Card Access Example

A major application of NFC is using NFC enabled phones to perform payments at point of sale registers [1]. This is done through tag emulation mode, and done through systems such as Google Wallet [2]. NFC can be used to set up cell phone configuration settings, pairing audio equipment, medical healthcare through patient ID and record linking, inventory management, or key fobs [2].

One specific NFC application is the ORCA cards used in Seattle [1]. The ORCA card system was launched in 2009, and consolidates most payments for public transportation [1]. Transportation systems such as ferries, busses, and trains all use the same ORCA cards for easier and faster payments and ticketing [1]. The hardware used in the ORCA cards are the Mifare DESfire IC [1].

The standard method by the NFC forum for exchanging data is called the NFC Data Exchange Format, or NDEF [1]. NDEF is used across almost all of NFC devices and data transmissions [2]. The NDEF consists of records, which is a single piece of data following four types, and messages, which organize the data contained in one or more records [1]. A message is an array of NDEF records with a header and a payload [1]. Messages can be chained together to support a larger amount of data [1]. The four types of NDEF records are text, URI, smart poster, and signature [1]. Text records contain a simple string of text, which includes metadata indicating the language and encoding scheme [2]. URI records contain network addresses, where the target device or reader is expected to direct the user to the network address through a web browser [2]. Smart Poster records are used by NFC tags attached to a poster to provide more information, which may be a URI [2]. Finally, signatures provide information about the origins of the data contained in the NDEF record [2]. The availability of a standard in NDEF formatting provides the key advantage NFC has over RFID [2].

NFC uses the physical principle of inductive coupling for power transfer and communication from the initiator to the device [3]. This process works by the reader injecting an alternating current in the reader antenna, resulting in an alternating magnetic field through the reader loop antenna [3]. This magnetic field induces an alternating voltage across the tag antenna through magnetic induction [3]. The tag electronics hardware contains a rectifier to convert this alternating current to a DC voltage supply [3]. Due to the inductive nature of NFC in contrast to radiative wireless common in most wireless technologies, antennas are commonly comprised of a simple coil of wire [3]. An example of the NFC inductive coupling is shown in Figure 2.2.

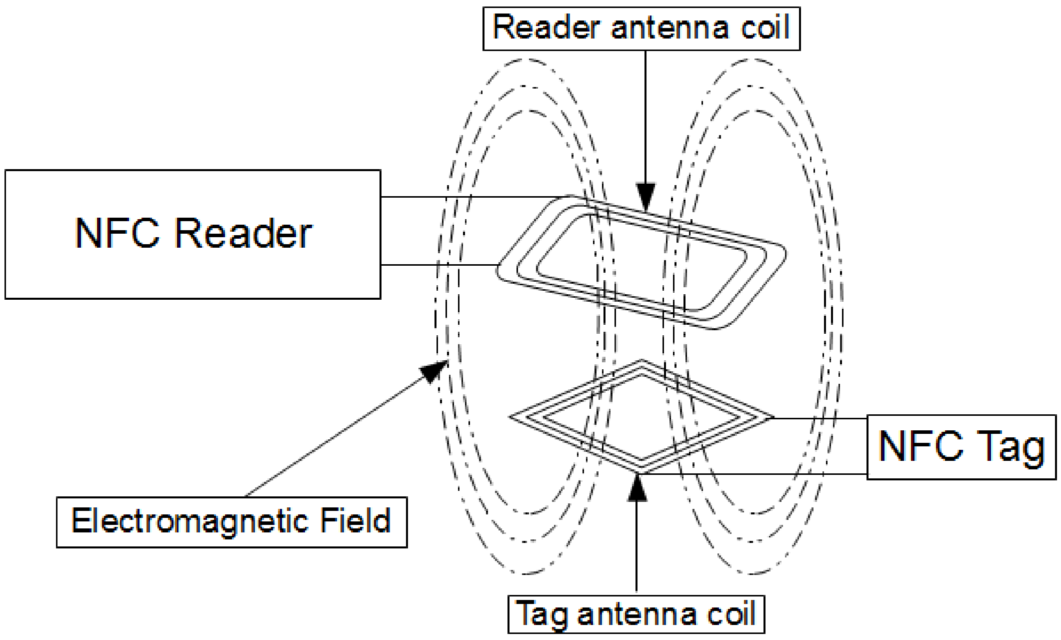


Figure 2.2: NFC Inductive Coupling

The reader modulates the amplitude of the alternating current to provide a means for transmitting data. This modulation method is called amplitude shift key, or ASK [3]. The change in amplitude corresponds to a data 0 or data 1 being transmitted. The target device will not have a power supply, and it will transmit data back to the reader through load modulation [3]. This works by the target varying the impedance of the antenna which corresponds to the data [3]. These changes in impedance appear as amplitude or phase changes across the initiator antenna [3]. The load modulation is achieved on a separate channel, with a carrier frequency of 848 kHz [3].

To convert the data to signals, three different encoding schemes may be used. They are Non-return to zero level (NRZ-L), Manchester, or Modified Miller [3]. The hardware project uses the ISO 14443 type A standard, which uses Modified Miller encoding from the initiator to the device [3]. The modulation method for the standard is on off keying modulation, or OOK, which is a specific case of ASK [3]. The load modulation data from the device to the initiator uses Manchester encoded data [3].

Chapter 3: Project Description

The purpose of the hardware project is to demonstrate the capabilities of NFC in a unique method. The battery-less, wireless sensor device showcases NFC advantages of not requiring on board power and wireless data transfer. The device provides environmental sensor data including temperature, pressure, relative humidity, and ambient light. The sensor device is read by an Android smartphone and this data is displayed on the application layout screen. The choice of sensors as the data source was made because it provides data which is not static and can be easily obtained and verified. The two main elements to the project are design and implementation. The design process includes wireless communication, power management, embedded firmware design, part selection, sensor integration, PCB trace antenna design, schematic design, and PCB layout. The implementation process involves assembling the PCB, tuning the antenna, creating the android application, and testing.

The system block diagram is shown in Figure 3.1. The two main hardware elements of the system design are the smartphone and the sensor device, or tag. The components on the smartphone are the application, or app, the NFC hardware, and the NFC antenna. These components are available and ready to use on the smartphone. The user application is the software running on the smartphone which handles communication with the tag. This is done by interfacing with the Android Operating System, or OS, and passing information through intents. The NFC hardware implements NFC standards for reading a tag through the NFC antenna. The antenna is often a loop of wire attached to the inside back of the smartphone case. For this project the smartphone model used was a Nexus 5X.

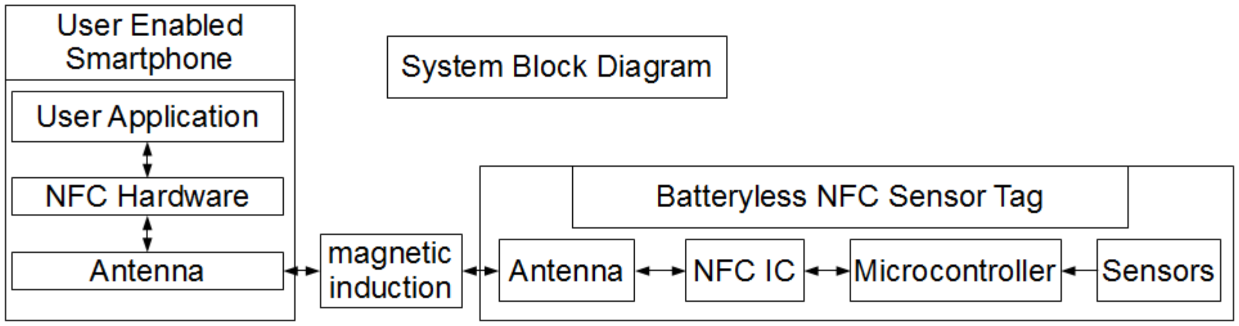


Figure 3.1: System Block Diagram

The Battery-less NFC Sensor Tag is custom hardware developed for the project. It contains a PCB trace antenna, an NFC IC, a microcontroller, and sensors. The PCB trace antenna is a square planar antenna designed to fit on a one inch by one inch square area. The NFC IC is an NXP dual port memory device, part number NT3H2211. The microcontroller is an 8-bit Microchip PIC16LF1709 which handles the sensor data collection and interfaces with the NFC IC for the data transfer. Finally, the sensors provide the data which is to be displayed on the smartphone. Figure 3.2 shows an example data transaction highlighted in a flowchart.

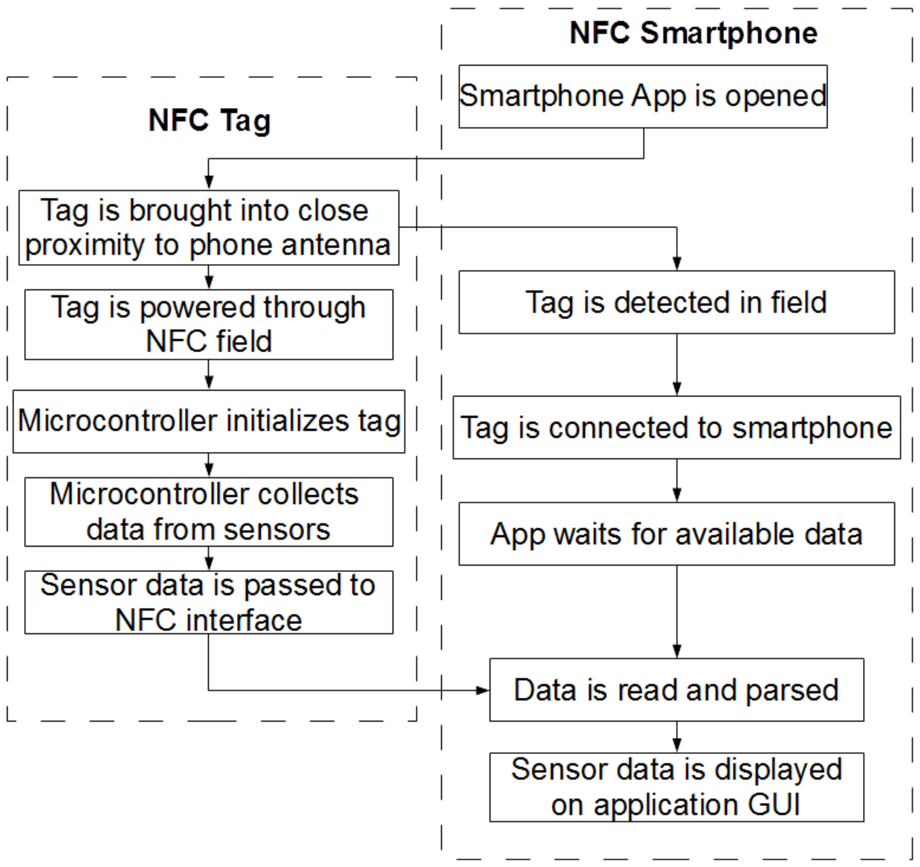

NFC Smartphone
Smartphone App is opened
NFC Tag
Tag is brought into close proximity to phone antenna
Tag is powered through NFC field
Microcontroller initializes tag
Microcontroller collects data from sensors
Sensor data is passed to NFC interface
Tag is detected in field
Tag is connected to smartphone
App waits for available data
Data is read and parsed
Sensor data is displayed on application GUI

Figure 3.2: Data Transaction Flowchart

Communication is initiated on the smartphone app by first enabling NFC hardware on the smartphone. The NFC hardware turns on the RF field and polls for an available NFC tag. Once the NFC tag is detected, an intent triggers the smartphone application to begin the NFC read method. The method then attempts to connect to the tag using the NFCa API. If successfully connected, and until data is available or the tag is disconnected, the app starts polling the NFC tag for data in SRAM. Meanwhile, after coming into the magnetic field, the NFC tag will convert the energy in the RF field to power on the microcontroller and supporting electronics. Upon power on, the microcontroller polls the RF field detect register of the NFC IC to make sure that the power is stable before beginning the power on sequence. Next the microcontroller sets the communication settings of the NFC IC to pass through mode, and sets the data direction to I2C to NFC to prevent the phone from accessing the data memory until the data is ready. The microcontroller then initializes the peripherals and external sensor hardware, and reads the sensor data. The last step for the microcontroller is to transfer the data to the NFC interface by filling the dual port memory SRAM buffer. Upon the SRAM buffer filling, the NFC IC automatically switches the data direction to NFC to I2C and the smartphone application can access the memory. Once the data is read, it is formatted and converted to a string and displayed in the corresponding text views.

The flowchart in Figure 3.3 outlines the design process for the system hardware project. The first step was part selection. The focus was placed on the NFC IC first, and then decisions were made for the rest of the system hardware based on the specifications of the NFC IC selected. After the major hardware components are chosen, the PCB trace antenna was designed. This gave a good idea of how much board space was needed to determine the size of the final PCB. Next a schematic was made which represents every component used and shows a detailed description of all the electrical connections between the components. From the schematic, a netlist was generated, which drove the layout of the PCB. The PCB layout gives a direct physical layout of all the components and copper traces for electrical connections. The PCB layout includes drawing the antenna designed earlier. After the PCB was designed, it was fabricated and ordered. The blank PCB was populated by hand soldering the components to the board, also known as PCB assembly. The next step was to develop the microcontroller firmware. After the firmware was completed, the software for the smartphone application and the graphical user interface was written. Finally, the system was tested for functionality and performance. Adjustments were made to fix any problems that were found. Any step along the design process was repeated after testing to fix any bugs discovered during testing.

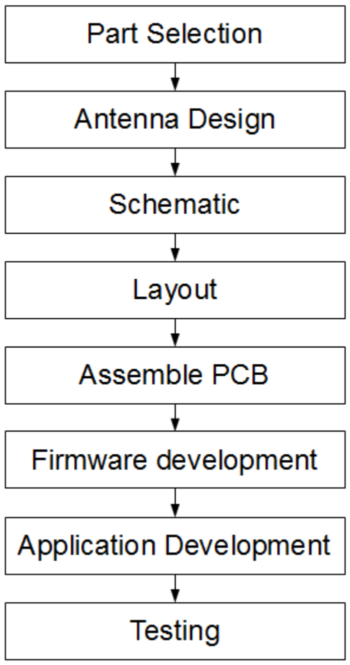


Figure 3.3: Design Process Flowchart

The NFC IC dictates what voltage levels are available and the maximum current draw of the system, and therefore is the central component in the design process. In considering the part selection of the embedded NFC hardware, there were two main options: a dual port memory device, or a system on a chip (SoC). Both options contain the hardware for implementing the NFC interface, as well as power harvesting capabilities to supply power to external hardware. The dual port memory has two separate interfaces for accessing the memory blocks. This allows an external serial device to communicate with an NFC device through the dual port memory. The system on a chip option includes the memory functional blocks as well as the NFC hardware, but also includes a microcontroller. This integrates the microcontroller into a single silicon wafer. Two specific options that were considered for the design are the TI RF420FRL152H NFC SoC, and the NXP NTAG I2C Plus dual port memory device. The TI RF430FRL152H supplies up to 1mA maximum output current at a regulated 3V from energy harvesting hardware [4]. The advantages of this option are a higher output voltage and a built-in microcontroller. The disadvantages are a lower output current and a higher cost per part. The NXP NTAG I2C Plus uses ISO 14443A Type 2 NFC standard, and can supply typical values of 2V and 5mA output from energy harvesting hardware [5]. The advantages of this option are higher output power supplied and simpler to implement. For this project the NXP NTAG I2C Plus was chosen as the NFC interface IC.

Once the NFC IC was selected a power budget was made. The design constraints are that the system current supply cannot exceed 5mA, and that hardware must have an operating voltage range around 2V. These requirements guided the selection of the rest of the hardware, starting with the microcontroller. The requirements for the microcontroller in addition to the power supply were at least 1 I^2C bus, an analog to digital converter, an internal operational amplifier, and an internal clock source. The selected part is a Microchip 8-bit microcontroller PIC16LF1709. This part has an operating voltage range from 1.8 to 3.6V [6].

Once the microcontroller was selected, the external sensors were selected. The information the sensors collect are temperature, air pressure, relative humidity, and light intensity. Each of the sensor devices operates at a range within the power budget. Apart from the light sensor which has an analog signal, each has an I^2C serial communication interface. To conserve power, only one digital sensor is active or awake at a time.

The communication bus for the digital devices is a single I^2C bus. In this configuration, the microcontroller is the master and the NFC IC, temperature, pressure, and humidity devices are the slave devices. Each slave device chosen has a seven-bit slave address, and each one needed a unique manufacturer address to prevent more than one device accessing the bus at a time.

I^2C has two open drain lines: the clock, or SCL, and the data, or SDA. Because they are open drain the data and clock lines need a pull up resistor to operate. To communicate, the master initiates communication with a start condition on the bus, and then writes the address of one of the slave devices. The slave devices monitor the bus for their respective address, and only respond to a request at their address.

Chapter 4: Hardware Implementation

The NFC communication on the sensor device side is handled by the NTAG I^2C Plus NFC hardware. The NTAG hardware options are configured by the microcontroller. For the project the device is run in pass-through mode for a simple one way data transfer. This mode is designed for data to be streamed 64 bytes at a time from either the NFC to I^2C direction or the I^2C to NFC direction. The data transfer is facilitated through an SRAM buffer. Once pass-through mode is enabled, only one interface may access the SRAM buffer at a time. Once an interface has filled the SRAM buffer with data, the data direction is automatically switched to allow the other interface to read the data. Once the data is read from the buffer the data is cleared and the original interface may write data to the buffer again. Any attempts to read the data from the restricted interface will be blocked and will result with a NACK or null. The process is sped up through a fast read and fast write commands, which will read or write the whole SRAM buffer in a single command. The slave address of the NTAG I^2C Plus is 0b1010101. [5]

The microcontroller selected operates at a voltage within the power requirements, from 1.8V to 3.6V. The microcontroller has a current draw of 32uA/MHz, allowing adjustment of the operating frequency to reduce the power. One of the two I^2C serial hardware units were used and configured for the master on the bus. The operational amplifier was used as the transimpedance amplifier for reading the photodiode. The internal voltage reference is set to 2.048V and is used to provide a stable reference level for the photodiode output measurement. There are 18 I/O pins available, and the IC uses an internal RC oscillator as the main clock source for the microcontroller as well as the serial interface. [6]

The temperature sensor selected is the TI TMP102 digital temperature sensor. It is configured in one shot mode to get a single temperature conversion before returning to sleep.

The temperature accuracy is plus or minus 2°C max within the operational temperature. The slave address of the TMP102 is 10010xx, and is hardware configurable depending on the connection of the A0 address select pin. The A0 address pin was connected to ground to set the address to 0b1001000. [7]

The pressure sensor selected is the Bosch BMP280 digital temperature and pressure sensor. It has an 8-pin LGA package, and a digital I^2C interface. The sensor is configured to operate in forced mode to get a single measurement. This makes the device go from the sleep state to the forced mode state, take a single measurement, and then return to sleep state to save power. The device can output several levels of pressure and temperature accuracy, and is configured with the lowest resolution to get the lowest current draw. The slave address of the IC can be configured by connecting an external pin to either power or ground, and in this case, was configured to have the address of 0b1110110. [8]

The humidity sensor selected was the TE Connectivity HTU2XY digital humidity sensor. The device operates at a voltage of 1.5 to 3.6V, with a maximum measuring current of 500uA. In addition to relative humidity, the sensor has a built-in temperature measurement. The sensor is configured in hold master mode, and takes a single measurement and ties up the I^2C data bus during a measurement. The slave address of the device is fixed at 0b1000000. [9]

The final sensor in the design is a Vishay TEMD6200 analog photodiode. This device was selected due to its simplicity and it has a spectral sensitivity similar to the human eye [10]. The device works by outputting a reverse light current which changes with incident light intensity levels. To measure this current, a transimpedance amplifier is used to convert the reverse diode current to voltage [11]. The circuit schematic of the transimpedance amplifier was simulated in LTSpice and is shown in Figure 4.1.

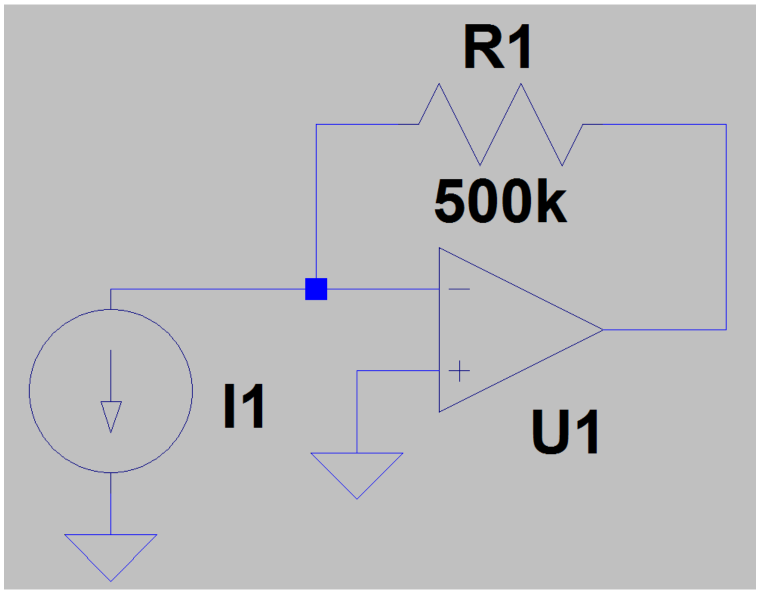


Figure 4.1: Transimpedance Amplifier Circuit

In the simulation, the photodiode is approximated as a current source [11]. The current I1 through the photodiode is the same as the current through the resistor R1. This results in a proportional voltage on the output of the operational amplifier, which is read by the analog to digital converter in the microcontroller. The photocurrent range is from 0.001uA to 5uA, and thus a 500k ohm resistor is used to get a voltage range which is within the 2V reference of the microcontroller [10].

Other components used in the hardware implementation include the Kingbright APTF1616SEEZGQBDC RGB LED. The LED is used for debugging purposes and as an indicator for the user. Surface mount ceramic decoupling capacitors were used in the 0402-package size. Surface mount chip resistors in the 0402-package size were also used where needed. Test point loops were included at important signal points for easily connecting an oscilloscope probe to the PCB.

After component selection, a bill of materials, or BOM is created. This is a list of all the electrical components used in the design. The final BOM is shown in Table 4.1.

Table 4.1: Bill of Materials

Ref Des	Description	Part Number	Value
U1	Light sensor	TEMD6200FX01	
U2	Microcontroller	PIC16LF1709-I/ML	
U3	Pressure sensor	BMP280	
U4	Temp sensor	TMP102AIDRLR	
U5	Humidity sensor	HPP845E031R5	
U6	NFC IC	NT3H2211W0FT1X	
D1	RGB LED	APTF1616SEEZGQBDC	
R1	Current Limiting R	RC1005F471CS	470 ohm
R2	Current Limiting R	RC1005F471CS	470 ohm
R3	Current Limiting R	RC1005F471CS	470 ohm
R4	Feedback R	RC0402FR-07499KL	500k ohm
R5	Pull Up R RD	RC1005J103CS	10k ohm
R6	Pull up R SCL	RC1005J103CS	10k ohm
R7	Pull Up R SDA	RC1005J103CS	10k ohm
R8	Jumper Vcc	RC1005J000CS	0 ohm
R9	Jumper Vout	RC1005J000CS	0 ohm
C1	Tuning Cap	GRM1555C1H240JA01D	23 pF
C2	Bypass Cap	GRM155R71C104KA88J	0.1 uF
C3	Bypass Cap	GRM155R71C104KA88J	0.1 uF
C4	Bypass Cap	GRM155R71C104KA88J	0.1 uF
C5	Bypass Cap	GRM155R71C224KA12D	0.2 uF

To determine the use for each configurable pin connection on the microcontroller, a pin planning process was used. Factors which must be weighed are: physical location of pin, capabilities of I/O pin, and configurable pins. Each pin and possible function were listed in a table. The final pinout of the microcontroller is shown in Table 4.2. This process is done before creating the schematic to prevent any incompatible pinouts.

Table 4.2: Microcontroller Pinout

QFN Pin #	IO	Planned Use	Description
16	RA0	ICSPDAT	Serial programming Data line
15	RA1	ICSPCLK	Serial programming Clock line
14	RA2	unused	
1	RA3	MCLR'	MCLR Master Clear reset pin
20	RA4	unused	
19	RA5	unused	
10	RB4	SDA	I2C Data
9	RB5	Photo ADC in	ADC input for photodiode
8	RB6	SCL	I2C Clock
7	RB7	NTAG FD input	Field detect pin input
13	RC0	LED3	Blue LED
12	RC1	LED2	Green LED
11	RC2	LED1	Red LED
4	RC3	OPA2OUT	Op amp output for photodiode circuit
3	RC4	unused	
2	RC5	unused	
5	RC6	OPA2IN-	Op amp inverting input for photodiode circuit
6	RC7	OPA2IN+	Op amp non-inverting input for photodiode circuit
18	Vdd	Vdd	positive supply
17	Vss	Vss	ground

The circuit schematic, shown in Figure 4.2, was created using the open source program KiCAD. Circuit components were made in a custom library by drawing a symbol and the pinouts for each part. The pinouts can be different from their physical location to improve readability of the schematic. For example, voltage inputs were placed on the top side of the IC symbol, and ground pins were placed on the bottom side of the IC symbol. The I^{2}C nets for the clock and data lines are labeled at several locations along the bus to prevent confusion between the two. Once the circuit schematic was complete, a netlist and association files were created to guide the PCB layout process.

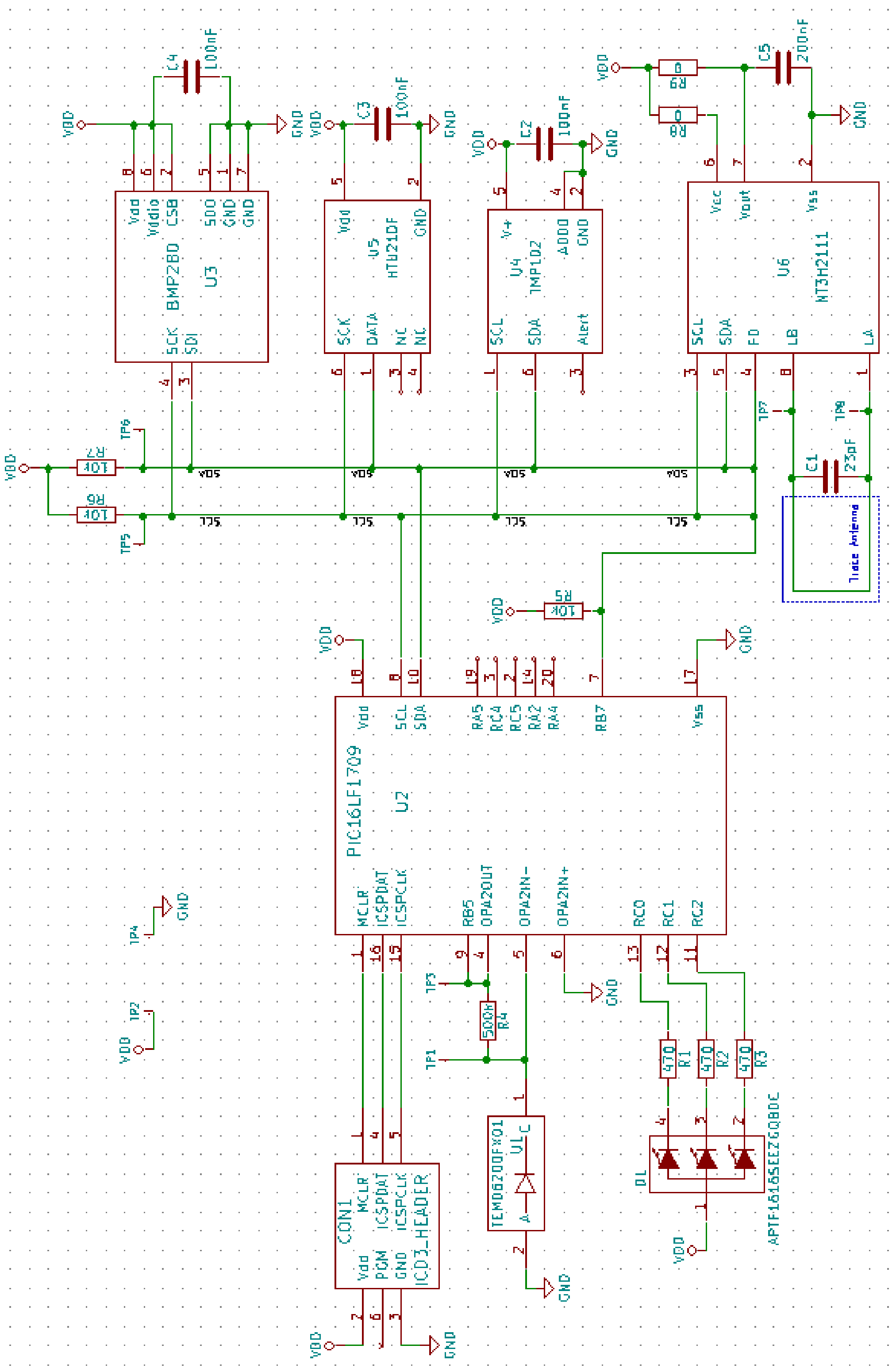


Figure 4.2: Schematic

The circuit connections for the NFC IC are shown in Figure 4.3. At least a 200nF capacitor is required for a stable power supply. The final tuning capacitor value of 23pF was used. The Vout pin supplies power to the rest of the system when proper inductive coupling occurs.

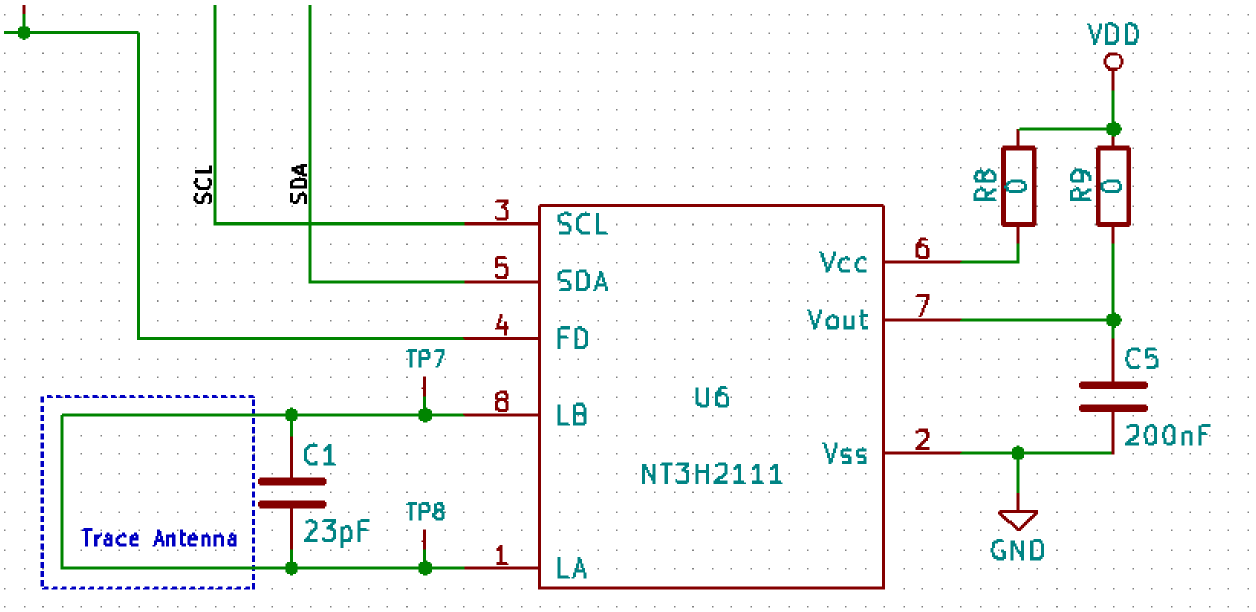


Figure 4.3: NFC IC Schematic

The antenna design motivation is to minimize antenna and board area while still being able to transfer energy and data via NFC. The choice was made to use a 1" square antenna. Ignoring series resistance, the resonant frequency of the antenna is given by [12]:

$$f = \frac{1}{2\pi\sqrt{LC}}$$

There are 3 antenna types typically used in NFC tags: circular antenna, spiral antenna, and square antenna [12]. The design of a square antenna was used to maximize the use of board space with a rectangular board. The inductance of the square antenna is given by [12]:

$$L = 2.34\mu_0 N^2 \frac{\frac{d_{out} + d_{in}}{2}}{1 + 2.75\frac{d_{out} - d_{in}}{d_{out} + d_{in}}}$$

Where d_{out} is the outer diameter of the square antenna, d_{in} is the inner diameter, N is the number of turns, and μ_0 is free space permeability. Using this equation, it is desirable to have as much inductance as possible, and as a rule of thumb to have at least 1uH [13]. Another factor to consider is that the larger surface area used, there will be more efficient coupling of energy [13].

Following the minimum design rules from the PCB manufacturer the trace width was chosen to be 0.2mm, and the trace clearance to be 0.2mm [14]. To calculate the inductance for a 1" board, the outer diameter d_{out} is to be 24.6mm by subtracting 0.4mm clearance. Next the number of turns was adjusted until a high enough inductance was calculated. Using 6 turns, an inner diameter of 22.6mm was found and the inductance was calculated to be 2.24uH. This design was confirmed based on the inductance calculated. Another design aspect is to adjust the antenna resonant frequency by using a parallel tuning capacitor. The input capacitance of the NTAG I^2C Plus antenna pins is 50pF [5]. Additional parallel capacitance was needed to resonate at 13.56MHz, and so the design is also confirmed. If a lower inductance is used, it would be more difficult to tune the antenna due to input capacitance of the device.

Some design rules should be followed during the layout process of the antenna. Due to the inductive nature of the antenna, all ground and components should be kept separate from the antenna. In addition, the antenna should be placed as close as possible to the input device, keeping the lead lengths as short as possible. [15]

The layout of the PCB was done in the KiCAD layout software. The design was fit into a 1"x2" rectangular area. The antenna design was copied onto one half of the PCB, and the component circuitry was put on the other half. The design is a 2-layer board, with parts only on the top side. Only 2 layers were needed due to the relatively low number of connections, and this was done to simplify debugging and the solder and assembly process.

The PCB layout began by copying the recommended footprint for each component. Next rough physical placement of the components was made, with an attempt to make connections as short as possible, especially between the NTAG I^2C Plus and the NFC antenna. A rat's nest was generated from the netlist, which showed connections which needed to be made between pins. Next copper connections were made, avoiding right angles and using as few vias as possible.

To have a physical interface for programming, a tag-connect programming connector was used. It allows for a connector-less layout because the tag-connect programmer cable contains a plug of nails. On the PCB, only circular exposed pads and three alignment holes are required. To connect the programmer to the PCB, the tag connect alignment posts are inserted into the PCB and the spring-loaded plug of nails makes connections for each programming pin. The recommended tag-connect layout was followed during the PCB design process. Once the final layout is finished, a design rules check is made to check for any problems such as overlapping traces. The output Gerber files were generated by the KiCAD software and sent to the PCB manufacturer for fabrication of the boards. The top and bottom design files are shown in Figure 4.4 and Figure 4.5, respectively.

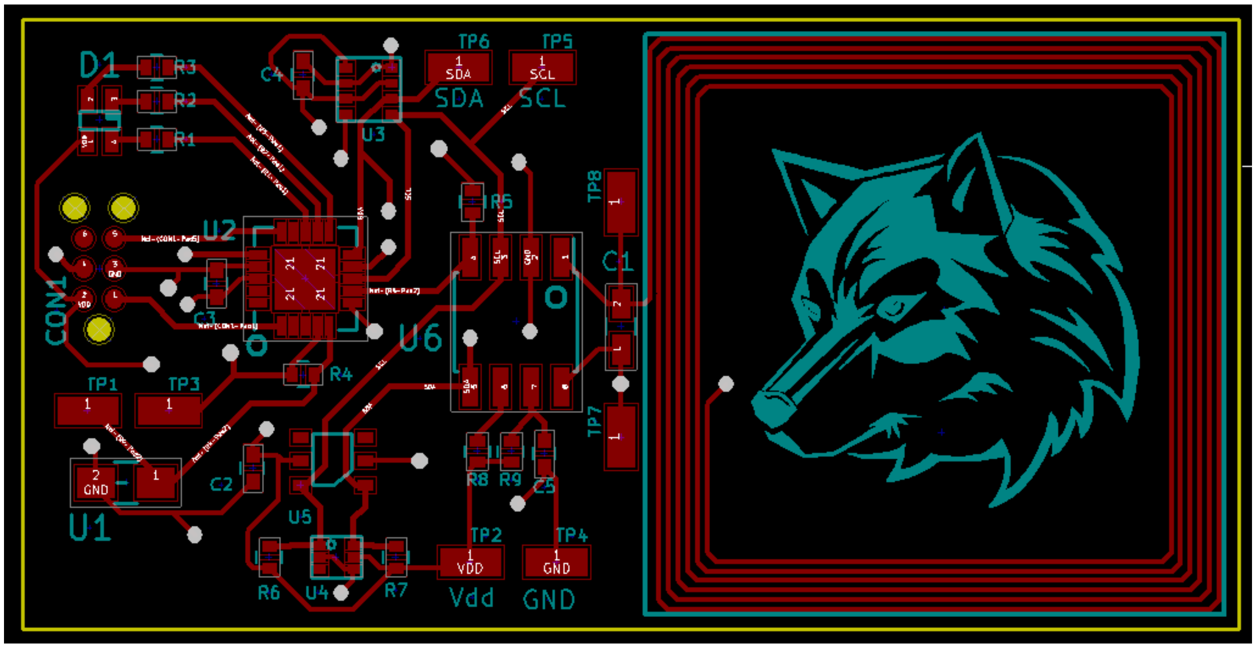


Figure 4.4: Top Side PCB Layout

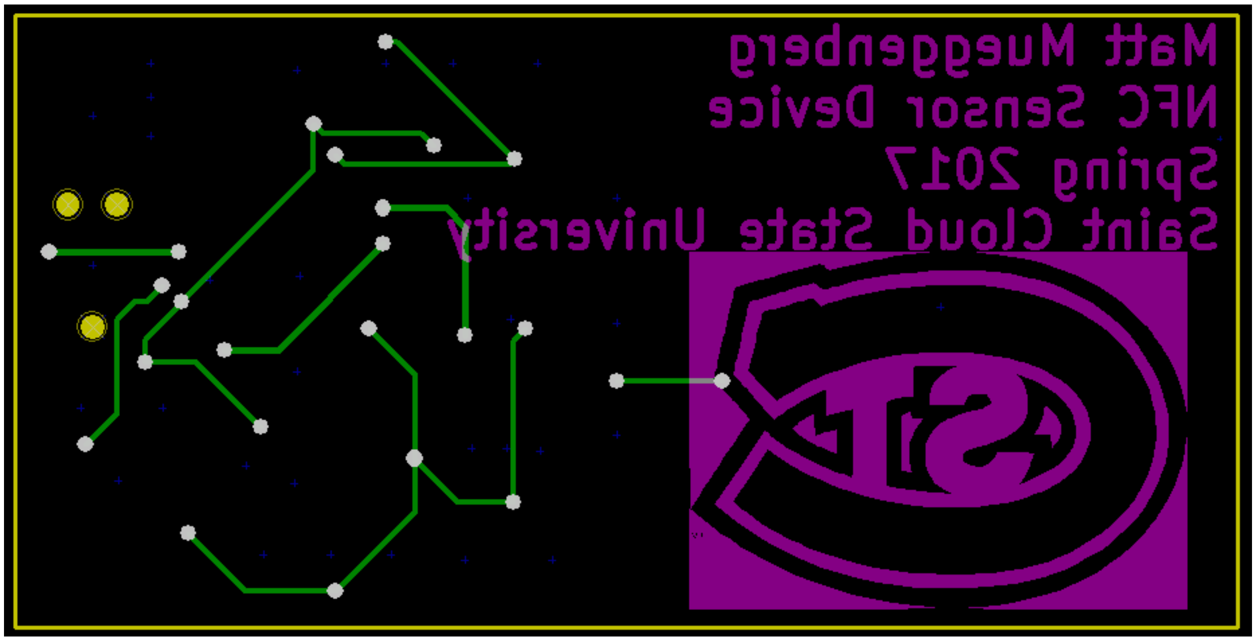


Figure 4.5: Bottom Side PCB Layout

The boards are rendered in 3D shown in Figure 4.6 and Figure 4.7.

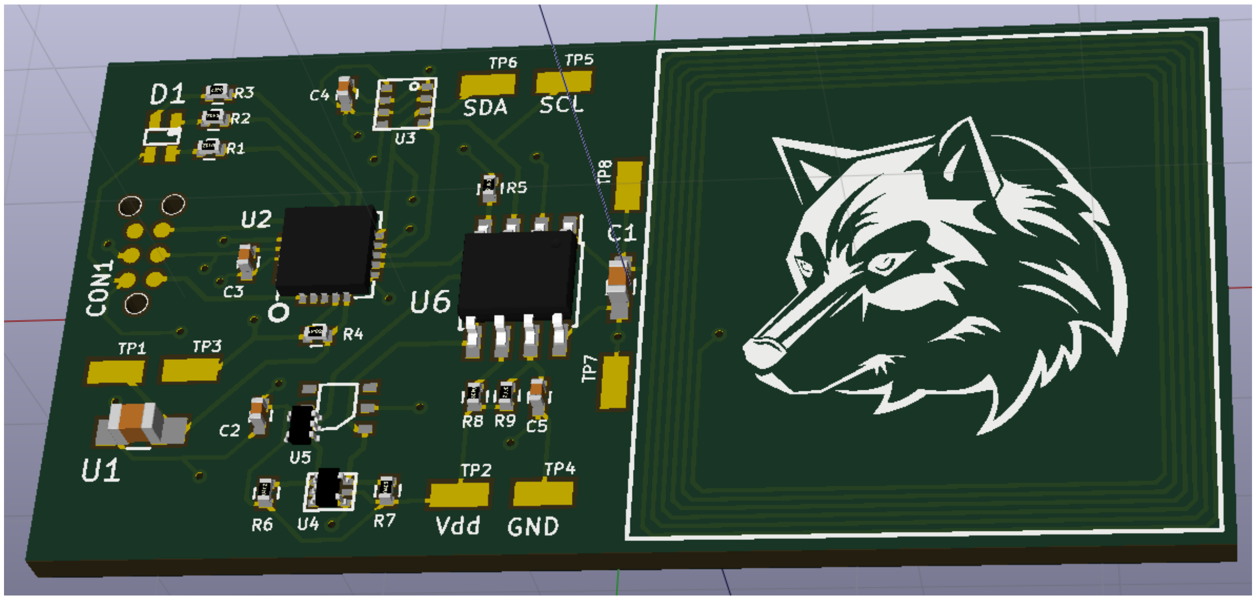


Figure 4.6: Top Side 3D Render

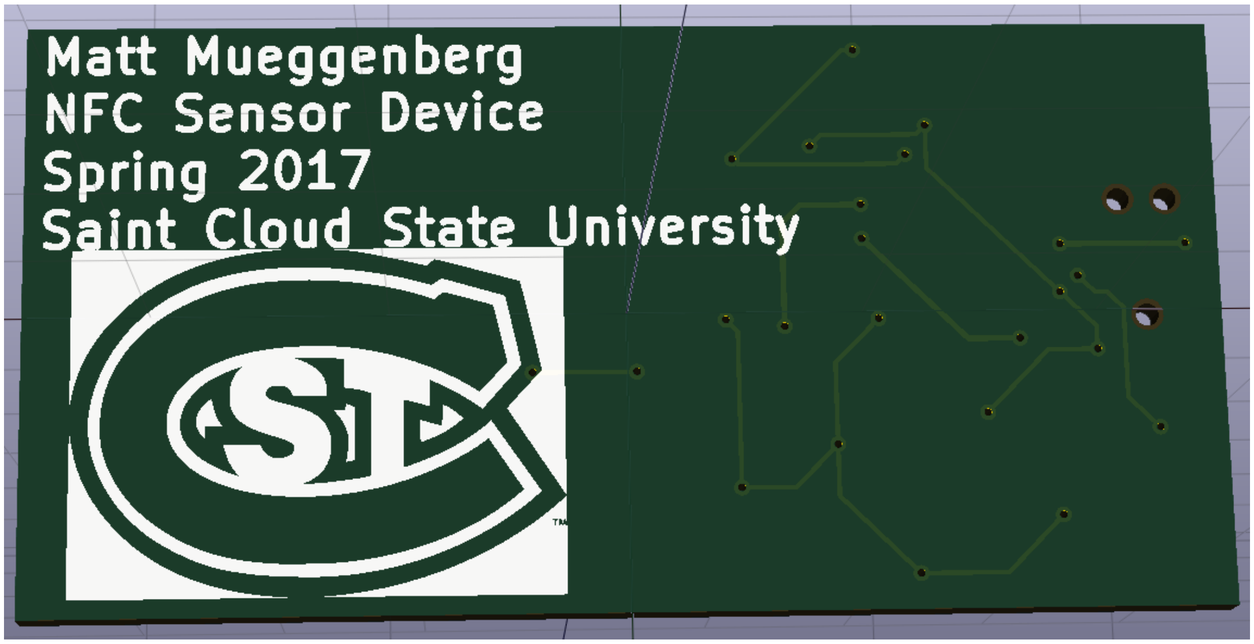


Figure 4.7: Bottom Side 3D Render

The assembly process was done by hand using a soldering iron and leaded solder. All the passive components were in a 0402 package. The design used QFN parts, which stands for quad flat with no lead. This was no problem for hand soldering when the lead runs on the side of the IC. This was done with the microcontroller and the humidity sensor. For the pressure sensor, the leads did not go onto the side of the package and so a different approach was taken. First the footprint pads were tinned with solder and coated in a generous amount of flux. Next the IC was placed onto the tinned pads. A hot air reflow tool was used to slowly heat up the part and melt the solder. To make sure that each pin was connected, the IC was poked with a pair of tweezers. The surface tension of the melted solder sprung the IC back into place, and a solid electrical connection was ensured.

The antenna tuning process was the last step in the hardware implementation. A network analyzer was used to determine the resonant frequency of the antenna. A custom-made loop probe was made from coaxial cable and the size of the loop was made to match the size of the device antenna. The loop probe was connected to the network analyzer in S11 mode to measure reflection [12]. Next the network analyzer was calibrated to account for the response of the cables and adapters connected to the port. The measurement was made by placing the loop probe onto the NFC antenna and sweeping the frequencies from 10 to 20MHz [12]. Figure 4.8 shows the loop probe and the device under test setup.

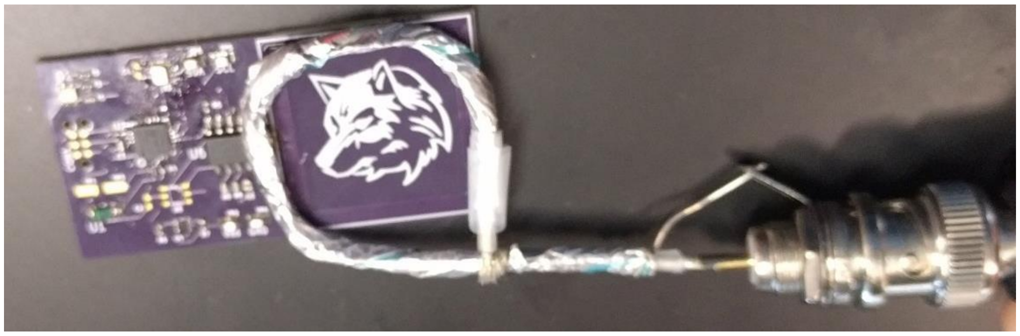


Figure 4.8: Loop Probe for Antenna Tuning

The initial measurement results are shown in Figure 4.9. The resonant frequency is shown by the marker at 16.586MHz. Next the tuning capacitor is adjusted and the antenna is measured again.

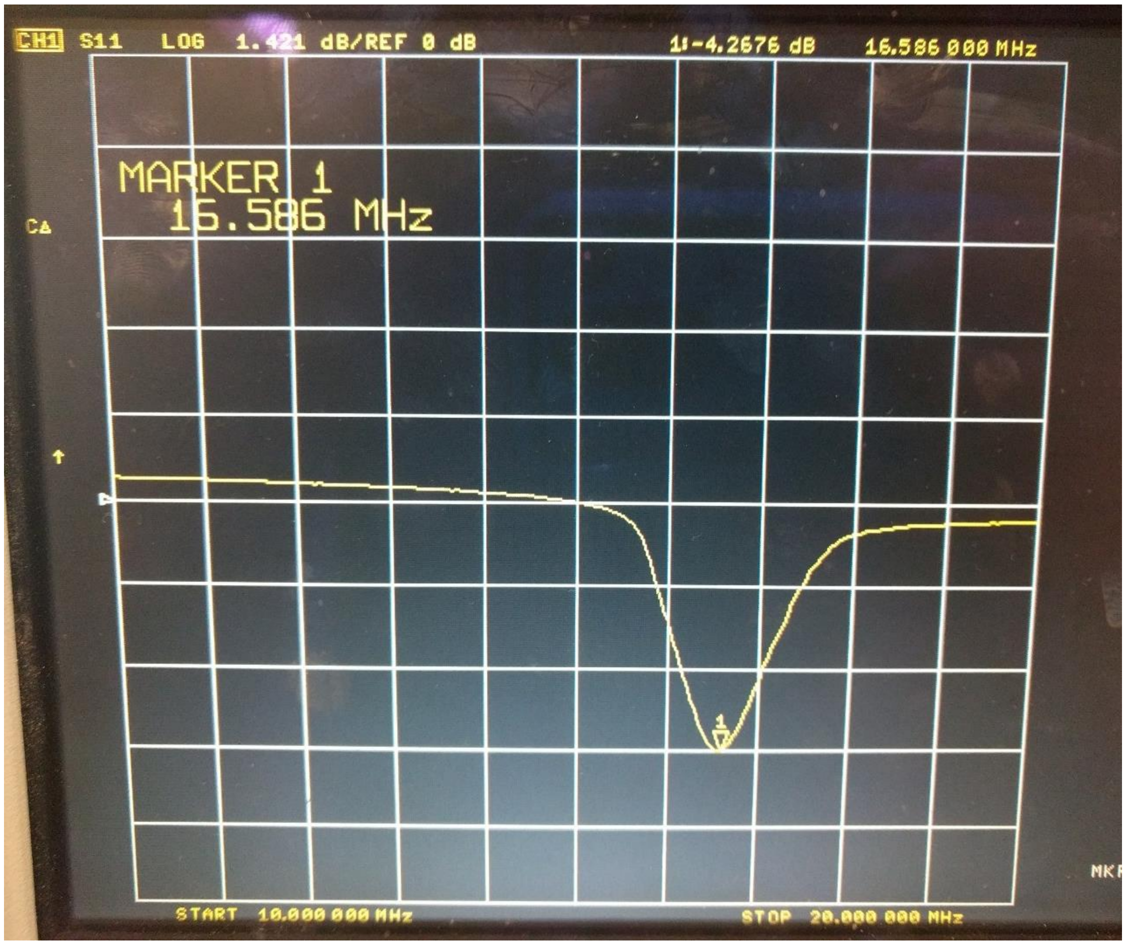


Figure 4.9: Untuned Antenna Response

After several iterations of measurement and tuning capacitor adjustment the resonant frequency of the antenna was measured to be very close to 13.56MHz. This was achieved using a tuning capacitance of 23pF. The output of the network analyzer in this hardware configuration is shown in Figure 4.10.

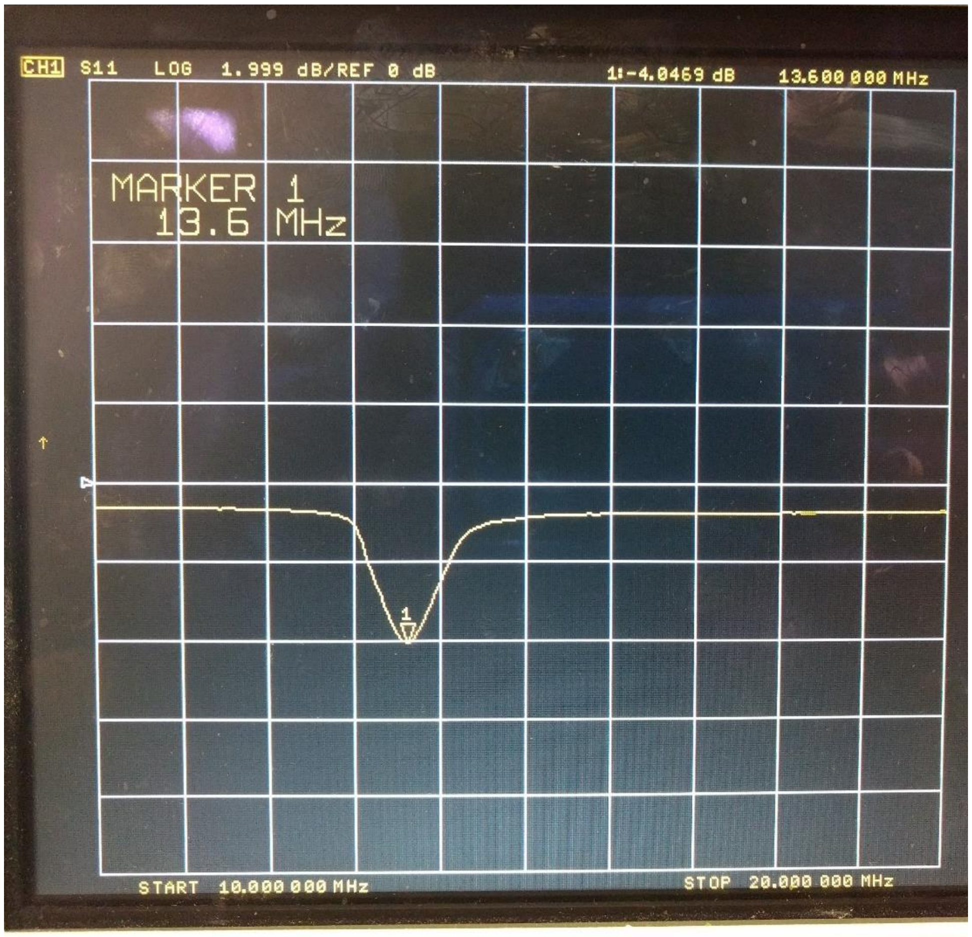


Figure 4.10: Tuned Antenna Response

Chapter 5: Firmware Design

The microchip development environment MPLAB X was used for the code design for the microcontroller. The firmware was written in the C language. The ICD3 programmer and debugger was used. The XC8 compiler V1.33 was used in free mode. The debugging process was difficult due to only having 1 hardware breakpoint available with the microcontroller used. To run debug mode or program the microcontroller, the board was connected to an external power supply.

The flowchart shown in Figure 5.1 shows the high-level functionality of the code. Upon power up, the microcontroller is initialized and waits for a stable voltage supply. Next the NFC interface is configured to prevent the smartphone from accessing the SRAM data buffer until the data is ready. The external sensors are each initialized, and the red LED is turned on to indicate that the power on sequence has finished. Next data is collected from each sensor and the data is stored into the SRAM buffer. The blue LED is turned on to indicate that the data is collected and available on the NFC interface to be read. The microcontroller waits for the FD pin to go low, indicating the smartphone has read the data and turns on the green LED indicating the transfer is completed.

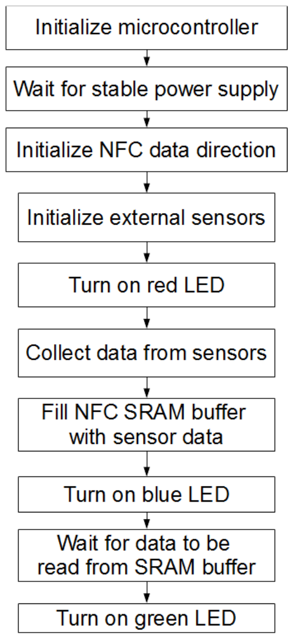
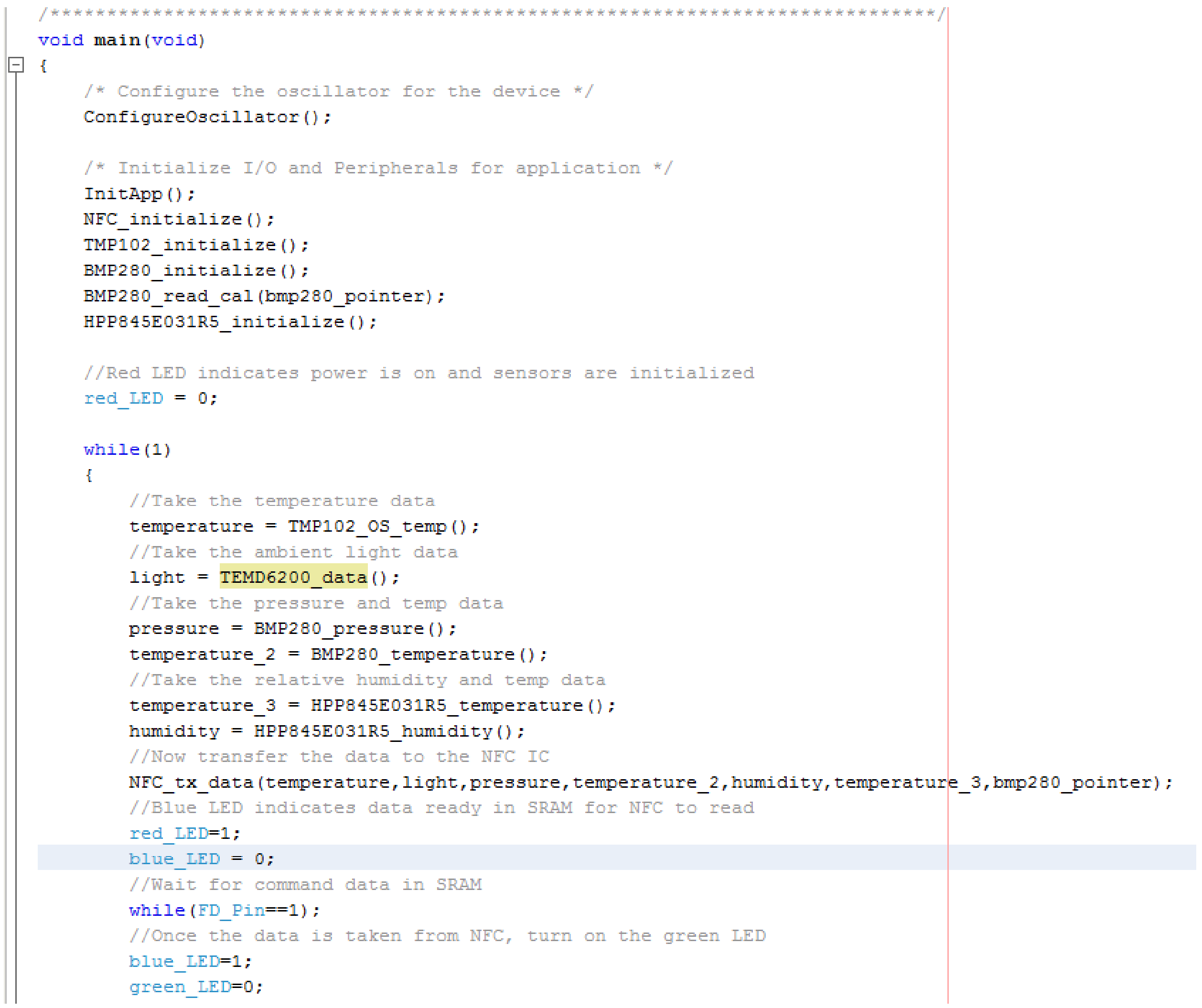

Initialize microcontroller
Wait for stable power supply
Initialize NFC data direction
Initialize external sensors
Turn on red LED
Collect data from sensors
Fill NFC SRAM buffer with sensor data
Turn on blue LED
Wait for data to be read from SRAM buffer
Turn on green LED

Figure 5.1: Firmware Flowchart

The main.c file is the central program file, containing the main function which runs at bootup. This file also contains variable declaration and is the central code for the device operation. The main function of the main.c file is the function which is run on power on. Figure 5.2 shows the code of the main.c function, which stitches together the important functions and drivers for the NFC interface and collecting the sensor data.



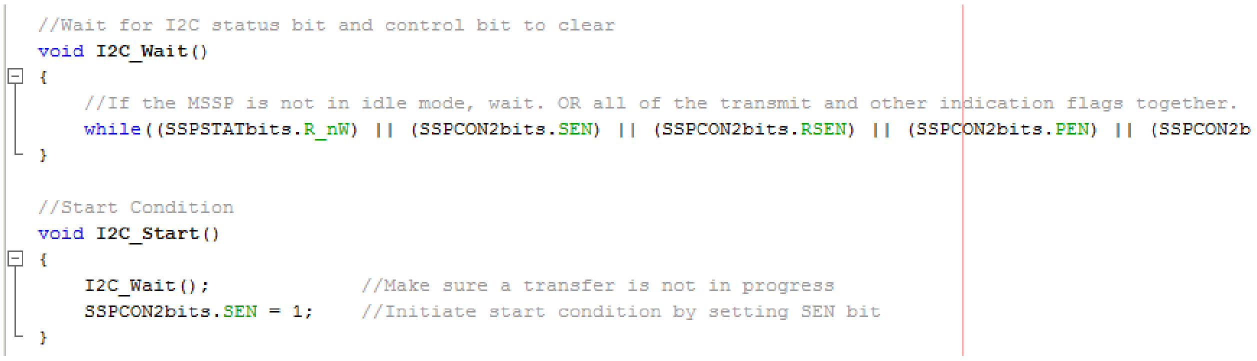
```
/******************************************************************************/
void main(void)
{
    /* Configure the oscillator for the device */
    ConfigureOscillator();

    /* Initialize I/O and Peripherals for application */
    InitApp();
    NFC_initialize();
    TMP102_initialize();
    BMP280_initialize();
    BMP280_read_cal(bmp280_pointer);
    HPP845E031R5_initialize();

    //Red LED indicates power is on and sensors are initialized
    red_LED = 0;

    while(1)
    {
        //Take the temperature data
        temperature = TMP102_OS_temp();
        //Take the ambient light data
        light = TEMD6200_data();
        //Take the pressure and temp data
        pressure = BMP280_pressure();
        temperature_2 = BMP280_temperature();
        //Take the relative humidity and temp data
        temperature_3 = HPP845E031R5_temperature();
        humidity = HPP845E031R5_humidity();
        //Now transfer the data to the NFC IC
        NFC_tx_data(temperature,light,pressure,temperature_2,humidity,temperature_3,bmp280_pointer);
        //Blue LED indicates data ready in SRAM for NFC to read
        red_LED=1;
        blue_LED = 0;
        //Wait for command data in SRAM
        while(FD_Pin==1);
        //Once the data is taken from NFC, turn on the green LED
        blue_LED=1;
        green_LED=0;
```

Figure 5.2: Main Function Source Code

The code is structured using two types of files, .h and .c. There are two header files, the system.h and the user.h files. The system.h file defines the system clock frequency and declares the oscillator configuration function. The user.h file defines all the initialization functions for the application and GPIO. It also defines the initializing and data reading functions for each sensor. It contains definitions for the NTAG I^{2}C Plus device, including initialization and configuration. Finally, it defines pin latches as green_LED, blue_LED, red_LED, and FD_Pin for easier code readability.

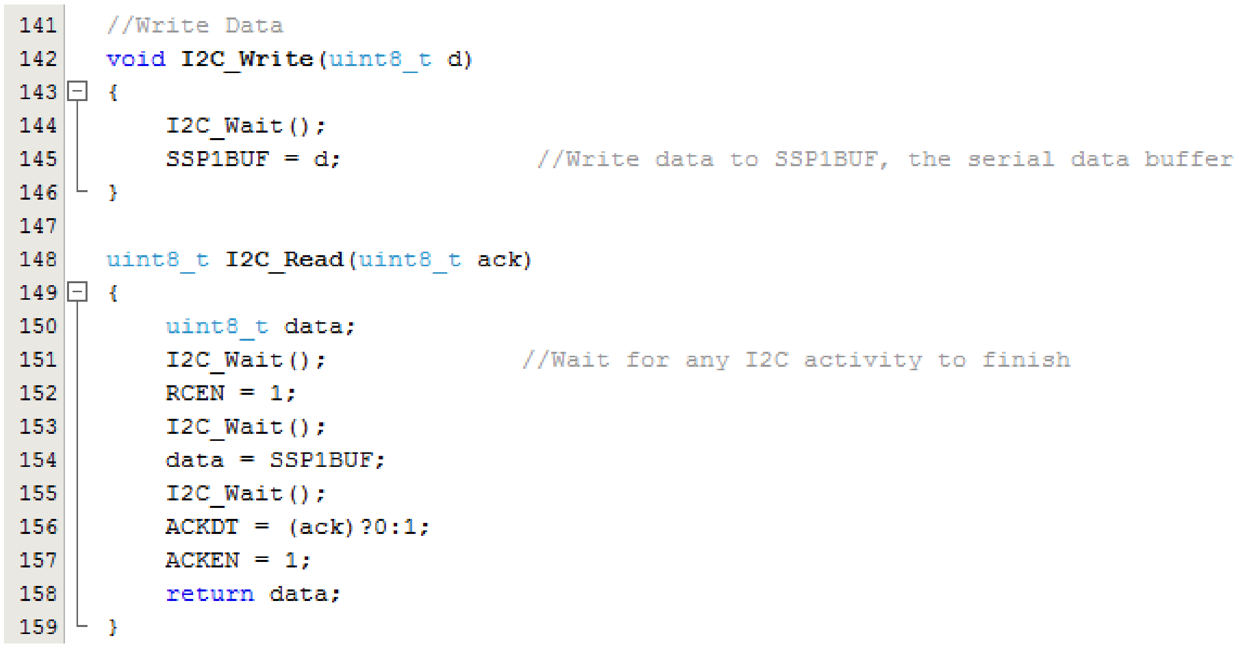
The source files are user.c, system.c, configuration_bits.c, and main.c. User.c contains the functions and drivers for the devices defined in the user.h header file. It also contains drivers for the I^{2}C communication in general. The drivers for the analog features are also contained in user.c. The system.c file contains the function to set up the oscillator. This function sets the operating frequency of the oscillator to 1MHz. The instruction clock is the clock frequency divided by 4, or 250kHz. The configuration_bits.c file has the device configuration pragmatic #pragma specifications which cannot be changed in the firmware.

The I^{2}C drivers were written to fulfill the basic set of requirements for I^{2}C communication. These include initialization, waiting, start condition, stop condition, reading, and writing. Figure 5.3 shows the initialize and waiting functions. The initialization function sets up the serial port and configures the microcontroller I^{2}C settings. The wait function polls status flags of the serial port control register until they are cleared, so that any remaining activity on the I^{2}C bus is completed. The start and stop functions simply initiate a start or stop condition on the bus by setting the appropriate control register. Finally, the read and write functions, shown in Figure 5.4, simply read or write data to the I^{2}C bus.



```
//Wait for I2C status bit and control bit to clear
void I2C_Wait()
{
    //If the MSSP is not in idle mode, wait. OR all of the transmit and other indication flags together.
    while((SSPSTATbits.R_nW) || (SSPCON2bits.SEN) || (SSPCON2bits.RSEN) || (SSPCON2bits.PEN) || (SSPCON2b
}

//Start Condition
void I2C_Start()
{
    I2C_Wait();             //Make sure a transfer is not in progress
    SSPCON2bits.SEN = 1;    //Initiate start condition by setting SEN bit
}
```

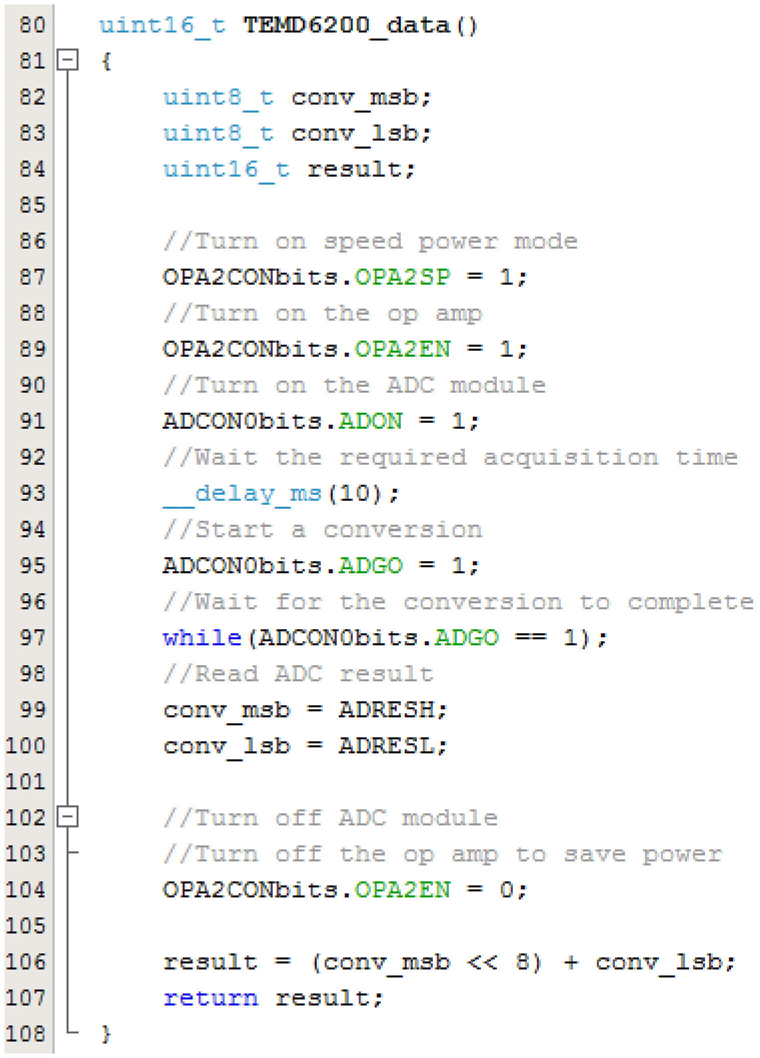
Figure 5.3: I^{2}C Wait and Start Functions



```
//Write Data
void I2C_Write(uint8_t d)
{
    I2C_Wait();
    SSP1BUF = d;                //Write data to SSP1BUF, the serial data buffer
}

uint8_t I2C_Read(uint8_t ack)
{
    uint8_t data;
    I2C_Wait();                //Wait for any I2C activity to finish
    RCEN = 1;
    I2C_Wait();
    data = SSP1BUF;
    I2C_Wait();
    ACKDT = (ack)?0:1;
    ACKEN = 1;
    return data;
}
```

Figure 5.4: I^{2}C Read and Write Functions

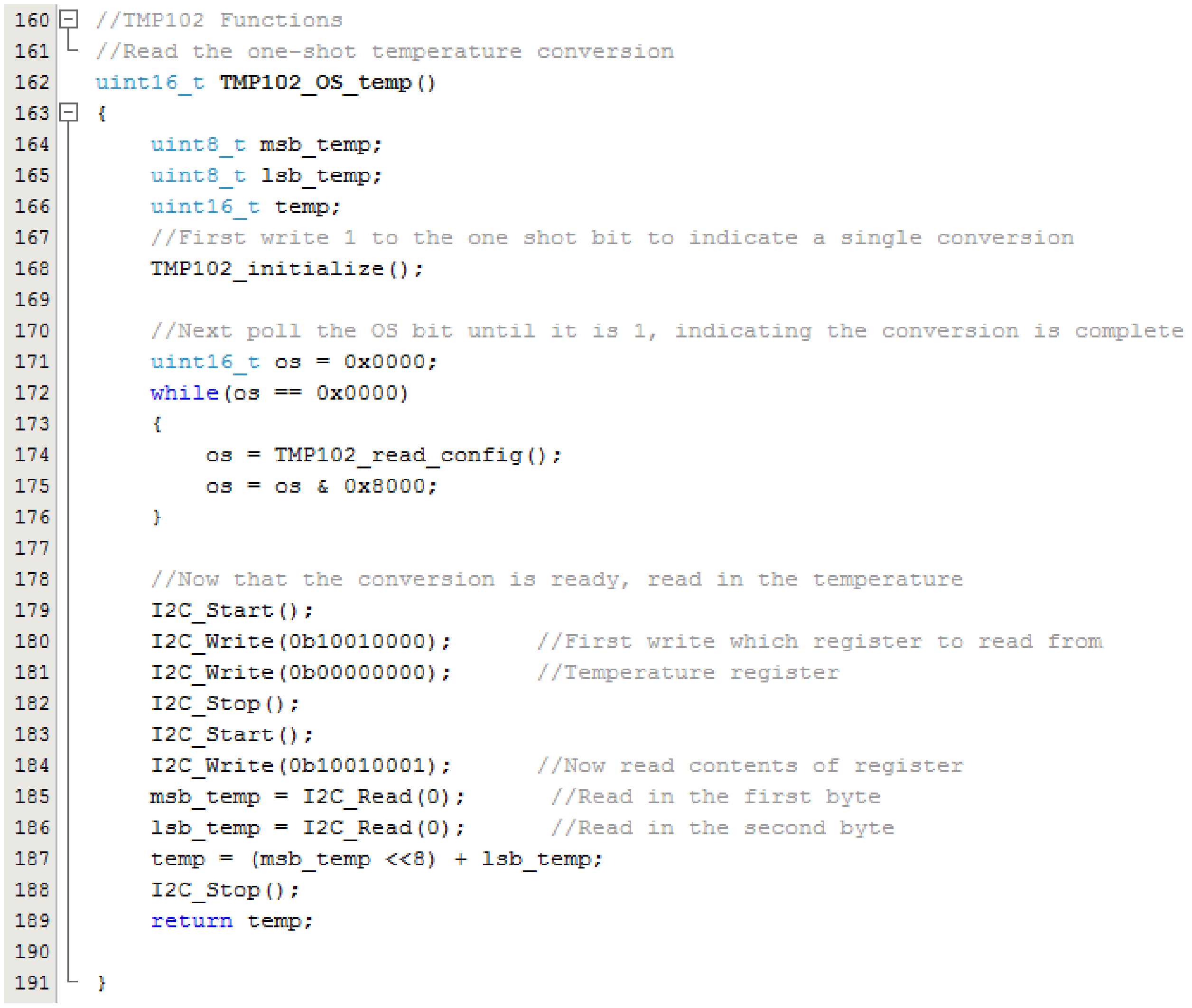
The functions for the TEMD6200 include turning on the op amp and the voltage reference. The first step is to initialize the analog components of the microcontroller which include the ADC input pin, the ADC voltage reference, and the operational amplifier. Once this step is completed, taking a voltage measurement is done using the TEMD6200_data() function, shown in Figure 5.5. The function turns on the op amp and ADC module, and then does an ADC conversion. The ADC module is then turned off and the result is returned.



```
uint16_t TEMD6200_data()
{
    uint8_t conv_msb;
    uint8_t conv_lsb;
    uint16_t result;

    //Turn on speed power mode
    OPA2CONbits.OPA2SP = 1;
    //Turn on the op amp
    OPA2CONbits.OPA2EN = 1;
    //Turn on the ADC module
    ADCON0bits.ADON = 1;
    //Wait the required acquisition time
    __delay_ms(10);
    //Start a conversion
    ADCON0bits.ADGO = 1;
    //Wait for the conversion to complete
    while(ADCON0bits.ADGO == 1);
    //Read ADC result
    conv_msb = ADRESH;
    conv_lsb = ADRESL;

    //Turn off ADC module
    //Turn off the op amp to save power
    OPA2CONbits.OPA2EN = 0;

    result = (conv_msb << 8) + conv_lsb;
    return result;
}
```

Figure 5.5: Photodiode ADC Conversion Function

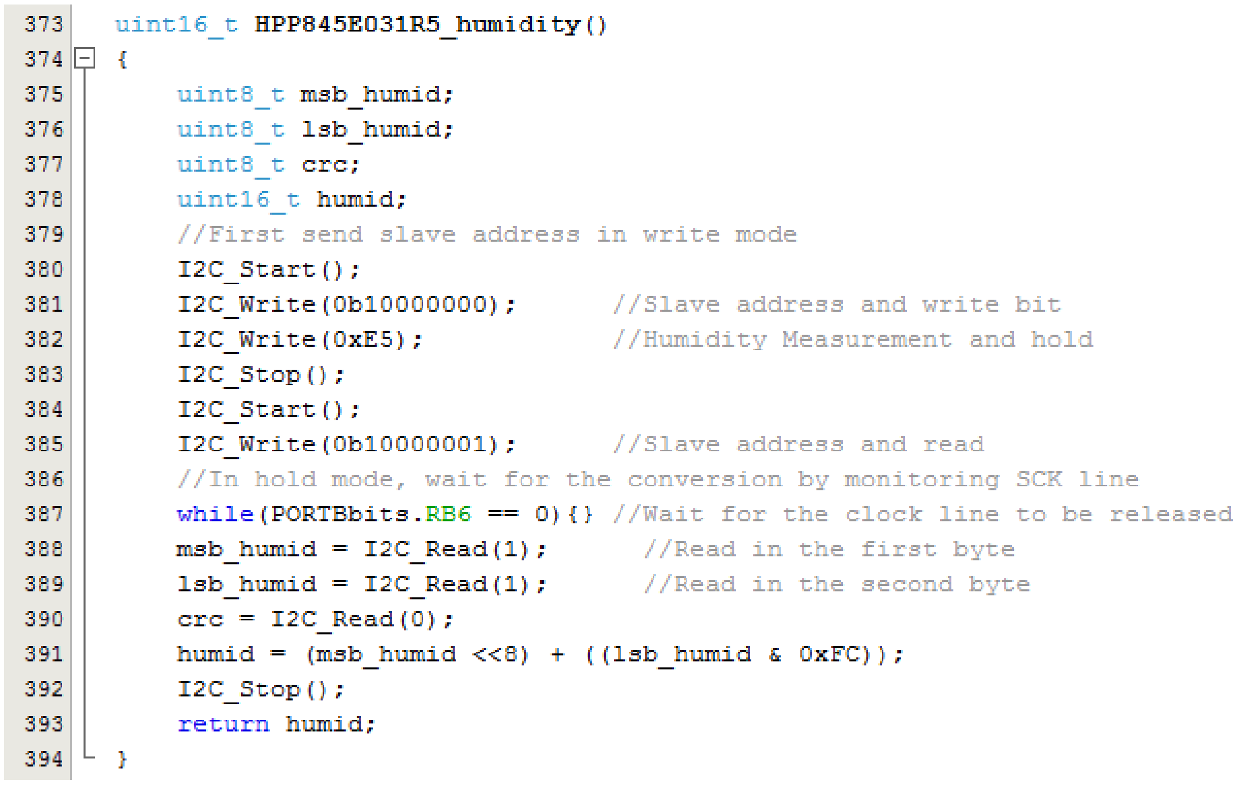
The TMP102 functions initialize the IC and do a one-shot reading of the temperature register. The initialize function simply writes to the configuration register to enable shutdown mode. The reading function turns on one shot mode, and then waits for the temperature data to be available. Once available, the temperature register is read and the data is returned. The TMP102_OS_temp() function is shown in Figure 5.6



```
//TMP102 Functions
//Read the one-shot temperature conversion
uint16_t TMP102_OS_temp()
{
    uint8_t msb_temp;
    uint8_t lsb_temp;
    uint16_t temp;
    //First write 1 to the one shot bit to indicate a single conversion
    TMP102_initialize();

    //Next poll the OS bit until it is 1, indicating the conversion is complete
    uint16_t os = 0x0000;
    while(os == 0x0000)
    {
        os = TMP102_read_config();
        os = os & 0x8000;
    }

    //Now that the conversion is ready, read in the temperature
    I2C_Start();
    I2C_Write(0b10010000);      //First write which register to read from
    I2C_Write(0b00000000);      //Temperature register
    I2C_Stop();
    I2C_Start();
    I2C_Write(0b10010001);      //Now read contents of register
    msb_temp = I2C_Read(0);      //Read in the first byte
    lsb_temp = I2C_Read(0);      //Read in the second byte
    temp = (msb_temp <<8) + lsb_temp;
    I2C_Stop();
    return temp;

}
```

Figure 5.6: Temperature Sensor Read Temperature Function

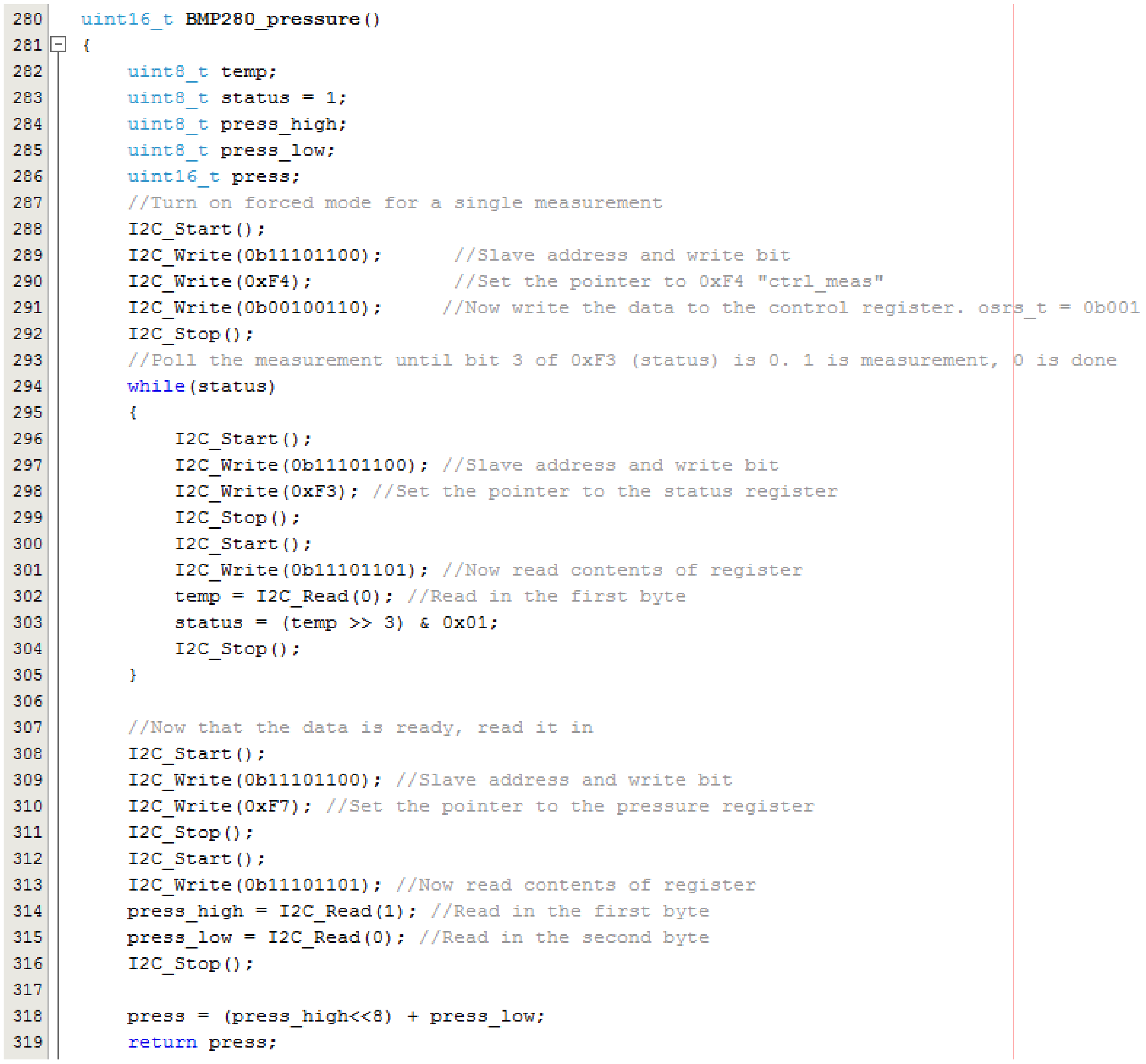
The HTU21D humidity sensor functions include an initialization, humidity reading and temperature reading. The initialize function simply resets the IC, which is recommended to do at the start up by the datasheet [9]. The humidity function initializes a single humidity conversion and then monitors the clock line until it is released, indicating the conversion is complete. Next the humidity data is read and returned. The temperature is read in the same manner as the humidity, with the only difference being a single temperature conversion is requested instead of humidity. Figure 5.7 shows the function for the HTU21D humidity reading.



```
uint16_t HPP845E031R5_humidity()
{
    uint8_t msb_humid;
    uint8_t lsb_humid;
    uint8_t crc;
    uint16_t humid;
    //First send slave address in write mode
    I2C_Start();
    I2C_Write(0b10000000);      //Slave address and write bit
    I2C_Write(0xE5);            //Humidity Measurement and hold
    I2C_Stop();
    I2C_Start();
    I2C_Write(0b10000001);      //Slave address and read
    //In hold mode, wait for the conversion by monitoring SCK line
    while(PORTBbits.RB6 == 0){} //Wait for the clock line to be released
    msb_humid = I2C_Read(1);      //Read in the first byte
    lsb_humid = I2C_Read(1);      //Read in the second byte
    crc = I2C_Read(0);
    humid = (msb_humid <<8) + ((lsb_humid & 0xFC));
    I2C_Stop();
    return humid;
}
```

Figure 5.7: Humidity Sensor Read Humidity Function

The BMP280 pressure sensor functions are initialize, read calibration register, read pressure, and read temperature. The initialize function resets the device and then writes configuration data to the control and configuration registers. The read calibration register reads the 26 bytes of calibration information stored on the device from factory calibration. This information is needed to calculate the pressure and the temperature values. The read pressure function turns on forced mode, and then polls the status register until it indicates the data is available. Once the data is available the 16-bit raw pressure data is read and returned. The temperature function works similarly to the pressure function, except that it skips setting the single measurement. Instead it requires that the pressure function has been called previously, and reads the temperature data which was generated along with the pressure data. Due to this, the temperature function can simply read the temperature result register and return the data. The read pressure function BMP280_pressure() is shown in Figure 5.8. [8]



```c
uint16_t BMP280_pressure()
{
    uint8_t temp;
    uint8_t status = 1;
    uint8_t press_high;
    uint8_t press_low;
    uint16_t press;
    //Turn on forced mode for a single measurement
    I2C_Start();
    I2C_Write(0b11101100);      //Slave address and write bit
    I2C_Write(0xF4);            //Set the pointer to 0xF4 "ctrl_meas"
    I2C_Write(0b00100110);     //Now write the data to the control register. osrs_t = 0b001
    I2C_Stop();
    //Poll the measurement until bit 3 of 0xF3 (status) is 0. 1 is measurement, 0 is done
    while(status)
    {
        I2C_Start();
        I2C_Write(0b11101100); //Slave address and write bit
        I2C_Write(0xF3); //Set the pointer to the status register
        I2C_Stop();
        I2C_Start();
        I2C_Write(0b11101101); //Now read contents of register
        temp = I2C_Read(0); //Read in the first byte
        status = (temp >> 3) & 0x01;
        I2C_Stop();
    }

    //Now that the data is ready, read it in
    I2C_Start();
    I2C_Write(0b11101100); //Slave address and write bit
    I2C_Write(0xF7); //Set the pointer to the pressure register
    I2C_Stop();
    I2C_Start();
    I2C_Write(0b11101101); //Now read contents of register
    press_high = I2C_Read(1); //Read in the first byte
    press_low = I2C_Read(0); //Read in the second byte
    I2C_Stop();

    press = (press_high<<8) + press_low;
    return press;
```

Figure 5.8: Pressure Sensor Read Pressure Function

The NTAG I^{2}C Plus provides the means for transmitting the data from the microcontroller to the NFC interface. To facilitate the transfer, the IC has a 64 byte SRAM data buffer. Upon power up, the buffer is accessed on a first come, first serve principle. Pass-through mode is used because it provides easy handshaking and fast read and fast write access. The data is transferred 64 bytes at a time through the SRAM buffer, which has an unlimited write endurance. The transfer direction is set which prevents access by one interface while the other interface is reading or writing to the buffer. Passthrough mode allows for automatically switching the access after the last read or write to the SRAM. [5]

The NFC settings for the NTAG I^{2}C Plus are set by first initializing the IC. First a function is run that checks that the NFC RF field is present and stable. This is accomplished by polling the RF_FIELD_PRESENT bit of the session register NS_REG at address 0x06. Once this bit is 1 it means that the field is present and stable. This step is used to prevent power supply problems during initializing the external sensors. Next the FD pin is configured. The FD pin is the field detect pin and it will be set to act as a semaphore for notifying when data is ready and when it is read. Next the data transfer direction is set to I^{2}C to NFC. This prevents the smartphone from accessing the SRAM before data is ready. Finally pass through mode is turned on. One these steps are taken, the NTAG I^{2}C Plus is now initialized set in pass-through mode. This process is highlighted in the initialize code developed shown in Figure 5.9. [5]

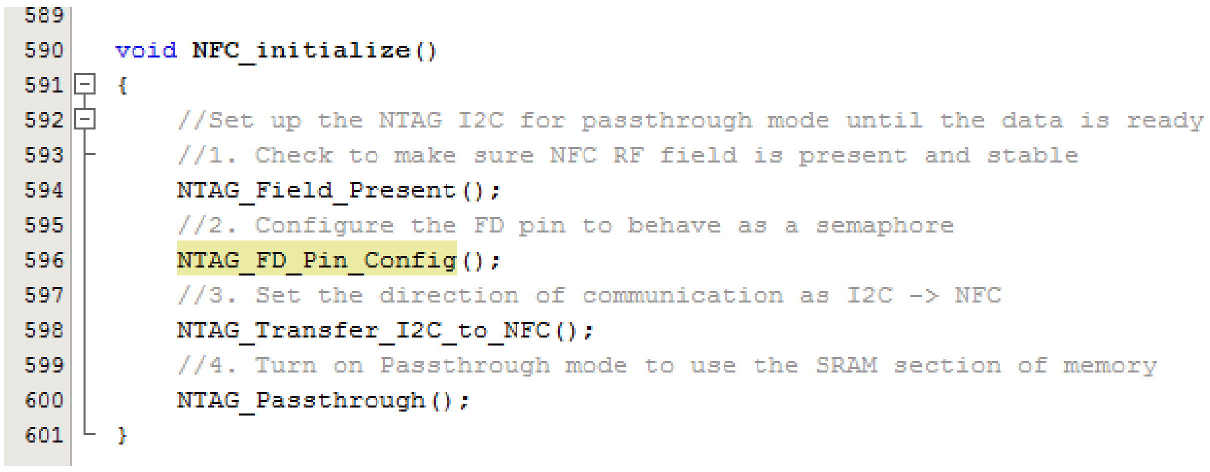
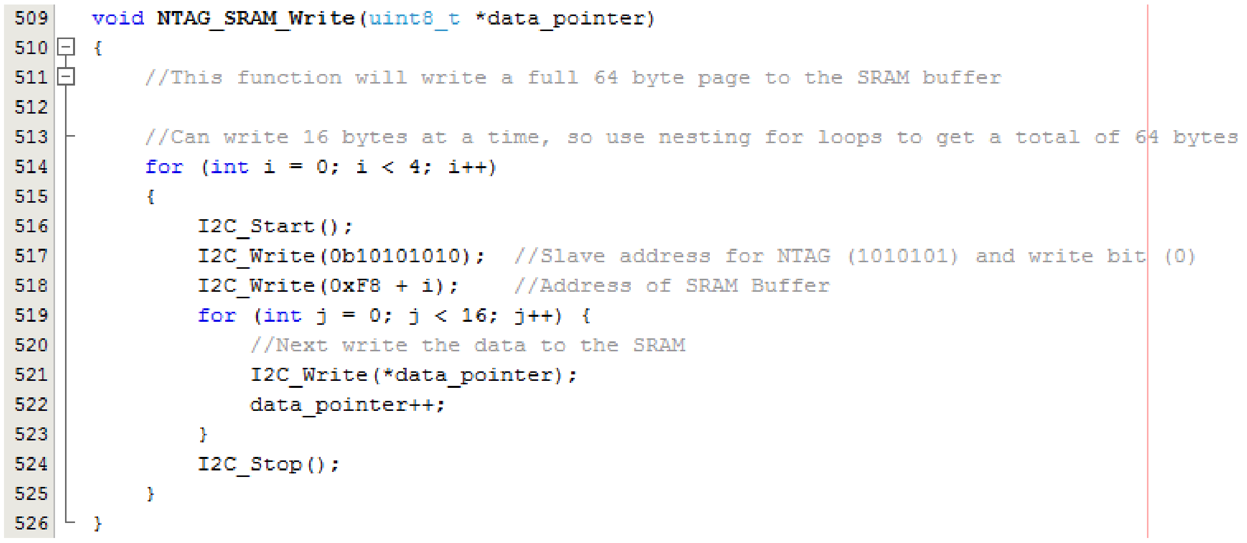

```

void NFC_initialize()
{
    //Set up the NTAG I2C for passthrough mode until the data is ready
    //1. Check to make sure NFC RF field is present and stable
    NTAG_Field_Present();
    //2. Configure the FD pin to behave as a semaphore
    NTAG_FD_Pin_Config();
    //3. Set the direction of communication as I2C -> NFC
    NTAG_Transfer_I2C_to_NFC();
    //4. Turn on Passthrough mode to use the SRAM section of memory
    NTAG_Passthrough();
}
```

Figure 5.9: NFC Initialize Function

To make an I^2C to NFC data transfer, first the above initialization must occur. Once pass-through direction is set and pass-through mode is turned on, data can be written to the SRAM buffer through the I^2C interface. During this time, any attempts to access the memory by the NFC interface results in a NACK, or not acknowledge. Once the I^2C interface completes writing to the last byte of the SRAM, the communication side is automatically switched. The NFC interface may now read the memory, and the I^2C interface is blocked from accessing the SRAM buffer. [5]

It was difficult to sort out the proper addresses for accessing the registers of the NTAG I^2C Plus due to having different addresses for the I^2C interface and the NFC interface. In addition, there are different addresses for the NFC interface depending on the operational mode and the IC memory type (2k or 1k). In this project the 2k memory version was used. The SRAM memory is addressed differently depending on the interface as well. From the I^2C interface, each SRAM memory address has 16 bytes of data to access, where the NFC interface each SRAM memory address has 4 bytes of data to access. The I^2C SRAM addresses used in the project are F8h to FBh, and the NFC SRAM addresses used were F0h to FFh. [5]

To fill the SRAM buffer, the microcontroller function combines the data to an array and passes the pointer of the array to the SRAM write function. The function iterates through nested for loops because there are 64 bytes to the buffer, and 16 bytes can be written at a time. The inner for loop writes the data, where the outer loop initializes the I^2C transfer with the SRAM address incrementing on each loop. This function is shown in Figure 5.10.



```
void NTAG_SRAM_Write(uint8_t *data_pointer)
{
    //This function will write a full 64 byte page to the SRAM buffer

    //Can write 16 bytes at a time, so use nesting for loops to get a total of 64 bytes
    for (int i = 0; i < 4; i++)
    {
        I2C_Start();
        I2C_Write(0b10101010);  //Slave address for NTAG (1010101) and write bit (0)
        I2C_Write(0xF8 + i);    //Address of SRAM Buffer
        for (int j = 0; j < 16; j++) {
            //Next write the data to the SRAM
            I2C_Write(*data_pointer);
            data_pointer++;
        }
        I2C_Stop();
    }
}
```

Figure 5.10: NFC SRAM Write Function

The firmware development was a significant portion of the design process and much of the code needed to be adjusted during the testing process. The development included writing drivers and functions for I^2C, as well as each digital external IC. Much of the testing and debugging process included stepping through the code and observing the digital I^2C signals on an oscilloscope to verify the communication and operation of the sensors and NFC IC.

Chapter 6: Software Design

The programming environment Android Studio was used for code design for the smartphone application. The programming languages used were Java and XML. Android studio was used with a USB C cable to program and debug the smartphone. The Android NFC Application Programming Interface, or API and Advanced NFC API for NFCa were used to access the library of NFC methods. There are two parts to the application, the Graphical User Interface, or GUI, and the logic which drives application.

The purpose of the application is to connect to the Wireless Battery-less Sensor Device, read the sensor data, and display it on the smartphone screen. An overview of the code is summarized in the flowchart in Figure 6.1. The program waits for an intent to connect to an available tag, issues the transceive command until data is available, reads and parses the data, and finally displays the data on the GUI.

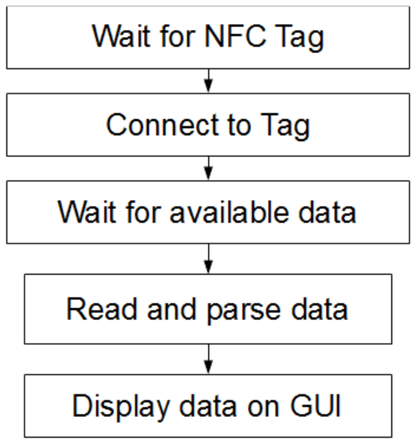
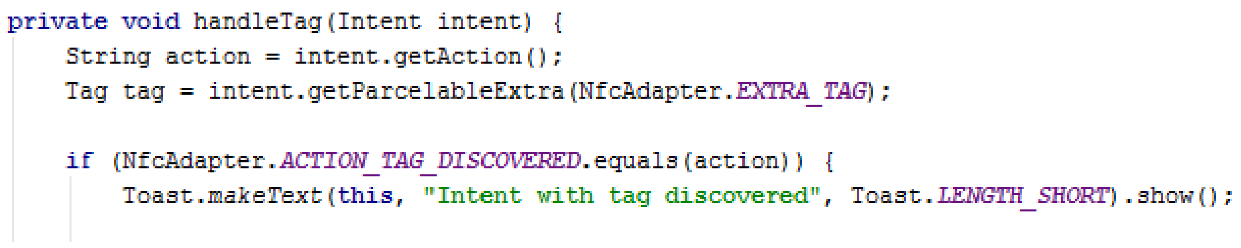


Figure 6.1: Application Flowchart

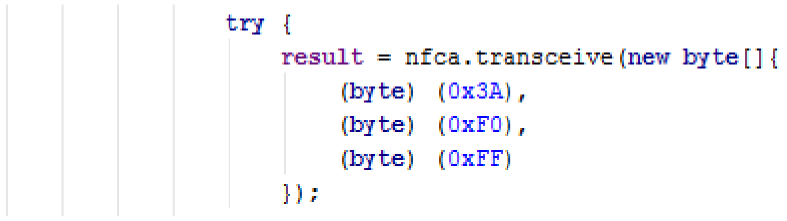
The Basic NFC API is the most common method for accessing the smartphone NFC hardware and doing basic memory read and write operations to NFC tags [16]. The majority of the Basic NFC API deals with NDEF and memory accesses. To activate the NFC hardware, the app uses permissions in the AndroidManifest.xml file [16]. An Android Intent is used to deliver tag information to the application [17]. From there, the tag is dispatched as a tag object from the intent. The handleTag() method shown in Figure 6.2 takes the intent and creates a Tag object.



```
private void handleTag(Intent intent) {
    String action = intent.getAction();
    Tag tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);

    if (NfcAdapter.ACTION_TAG_DISCOVERED.equals(action)) {
        Toast.makeText(this, "Intent with tag discovered", Toast.LENGTH_SHORT).show();
```

Figure 6.2: Handle Tag Method

The Advanced NFC API allows for specific operations on input and output beyond NDEF. The NfcA provides access to ISO 14443-3A properties and operations. This allows for a way to implement reading the SRAM buffer of the NTAG I^{2}C Plus. Important methods include: get(Tag tag), connect(), isConnected(), transceive(), and close(). The NfcA transceive() method is used to execute the NFC read operation of the SRAM. The transceive() method automatically calculates and appends the CRC to the data stream. The NFC command for fast read, as written in the NTAG I^{2}C Plus datasheet, is 3Ah. Figure 6.3 shows the code implementation for the transceive() method. The fast read command, followed by the starting and ending addresses of the SRAM are input to the transceive() method. The result object stores the data read out from the SRAM buffer. [18]

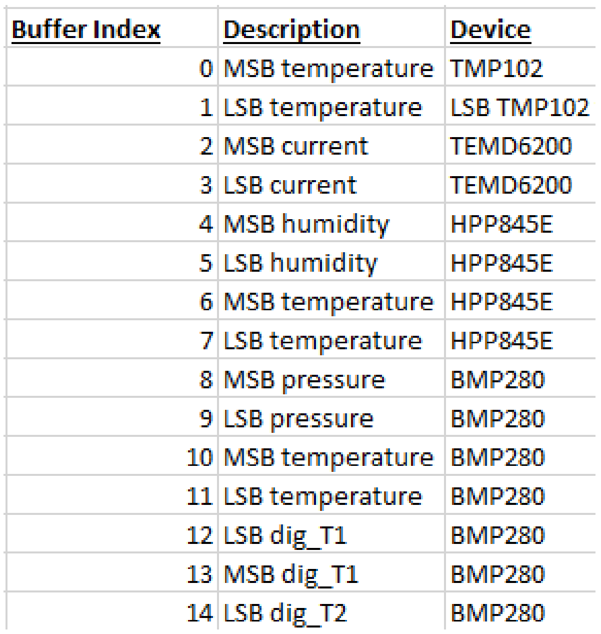


```
try {
    result = nfca.transceive(new byte[]{
        (byte) (0x3A),
        (byte) (0xF0),
        (byte) (0xFF)
    });
```

Figure 6.3: Transceive Method

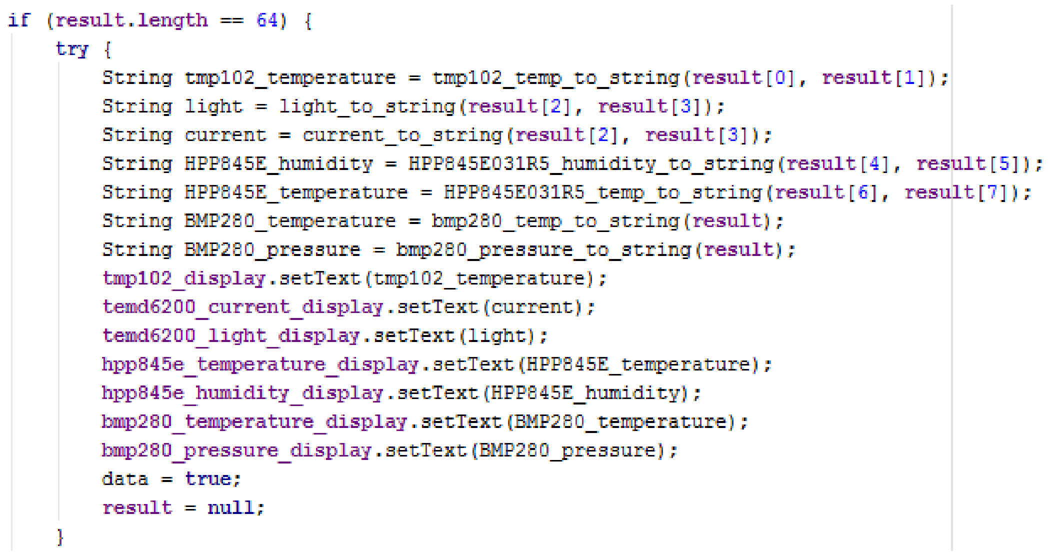
Once the data is read and stored in the result object buffer, it needs to be processed and parsed. A method was created for each data point to convert the raw data into the corresponding temperature, relative humidity, pressure, and light intensity value. To keep track of the data in the buffer, the table shown in Table 6.1 was created which organizes the buffer index for the data written from the microcontroller and read from the smartphone.

Table 6.1: Buffer Index Description



Buffer Index	Description	Device
0	MSB temperature	TMP102
1	LSB temperature	LSB TMP102
2	MSB current	TEMD6200
3	LSB current	TEMD6200
4	MSB humidity	HPP845E
5	LSB humidity	HPP845E
6	MSB temperature	HPP845E
7	LSB temperature	HPP845E
8	MSB pressure	BMP280
9	LSB pressure	BMP280
10	MSB temperature	BMP280
11	LSB temperature	BMP280
12	LSB dig_T1	BMP280
13	MSB dig_T1	BMP280
14	LSB dig_T2	BMP280

Figure 6.4 shows the code which passes the proper result buffer data to each parsing method, and subsequently passes the strings into their respective TextViews. The application first checks if the buffer is the proper length to prevent the code from hanging if the communication process was interrupted, resulting in only a partially filled result buffer.



```
if (result.length == 64) {
    try {
        String tmp102_temperature = tmp102_temp_to_string(result[0], result[1]);
        String light = light_to_string(result[2], result[3]);
        String current = current_to_string(result[2], result[3]);
        String HPP845E_humidity = HPP845E031R5_humidity_to_string(result[4], result[5]);
        String HPP845E_temperature = HPP845E031R5_temp_to_string(result[6], result[7]);
        String BMP280_temperature = bmp280_temp_to_string(result);
        String BMP280_pressure = bmp280_pressure_to_string(result);
        tmp102_display.setText(tmp102_temperature);
        temd6200_current_display.setText(current);
        temd6200_light_display.setText(light);
        hpp845e_temperature_display.setText(HPP845E_temperature);
        hpp845e_humidity_display.setText(HPP845E_humidity);
        bmp280_temperature_display.setText(BMP280_temperature);
        bmp280_pressure_display.setText(BMP280_pressure);
        data = true;
        result = null;
    }
```

Figure 6.4: Data Parsing Methods

The application layout, or GUI was designed using a constraint layout. This allows for objects in the layout to be located at set distances from each other. The layout was created using the Android Studio layout design tool. The layout itself simply comprised of several TextView objects for displaying data and data labels and headers. The blueprint for the main activity xml layout is shown in Figure 6.5

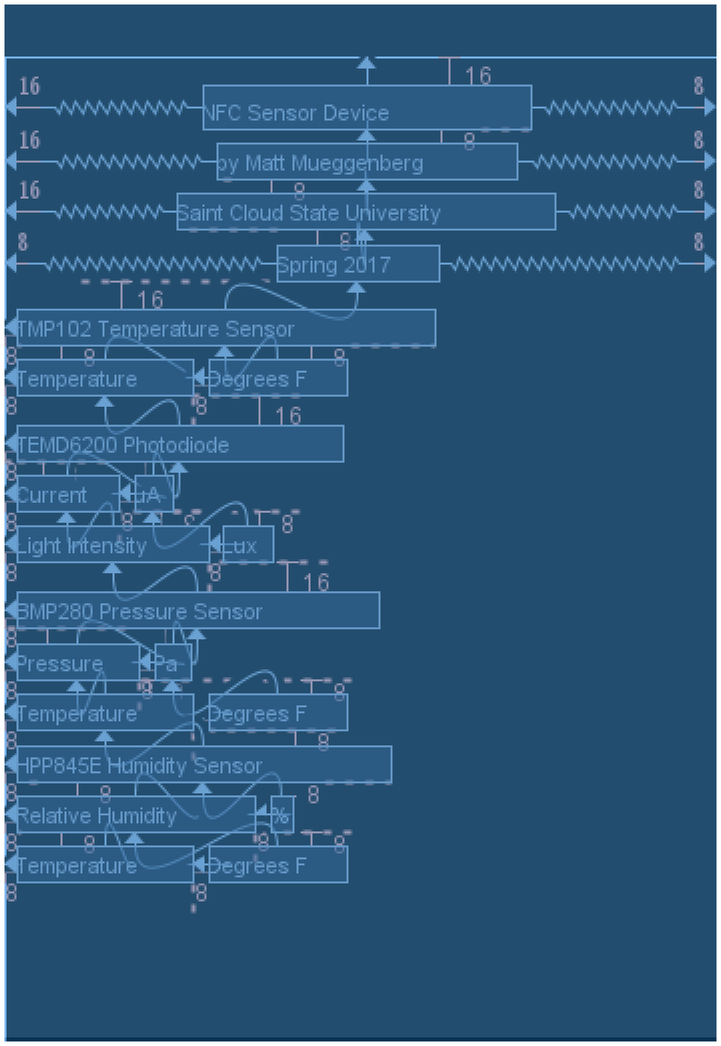


Figure 6.5: Application Layout Blueprint

Chapter 7: Results and Analysis

The result of the design is a single tag to smartphone application sensor data transfer. The wireless, battery-less NFC sensor device is populated and shown in Figure 7.1

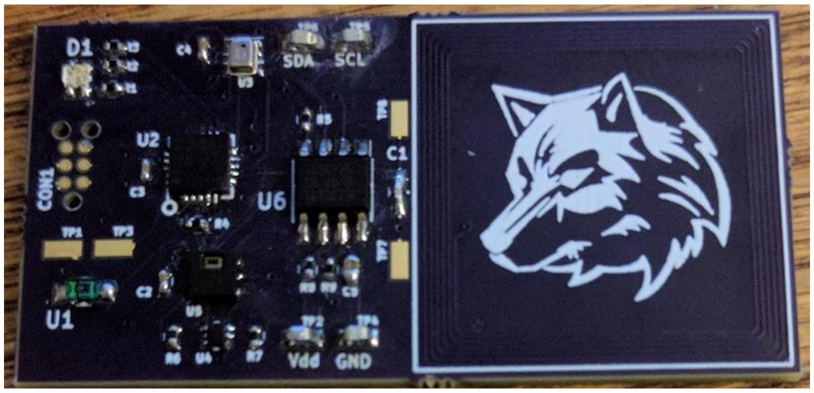


Figure 7.1: Populated PCB

A demonstration of the system operating is shown in Figure 7.2. For the best results the sensor device should be placed in direct or very close contact to the back side of the smartphone. The sensor device was not able to work when placed on a metal table or metal environment. The best results occur when the smartphone antenna is perpendicular to the sensor device antenna, however successful data transactions were recorded with the sensor antenna out of alignment, at times 45 degrees off axis or more.

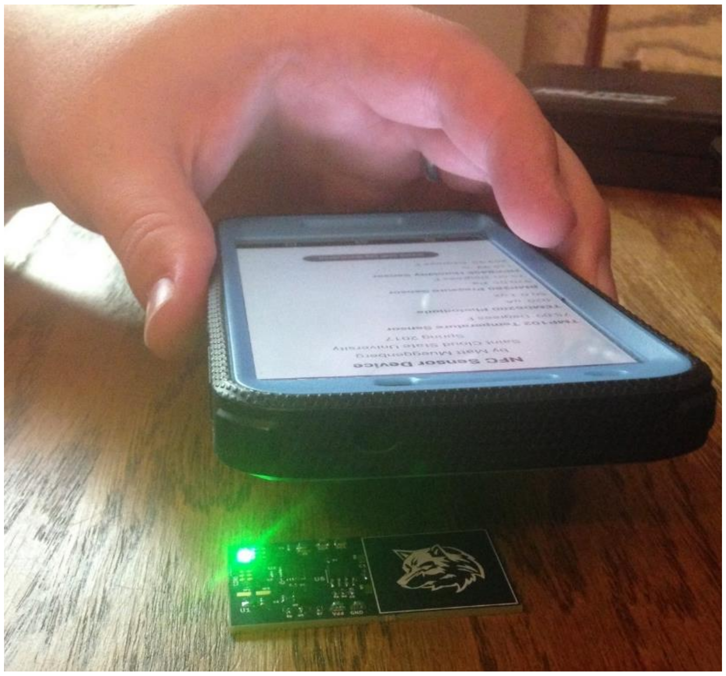
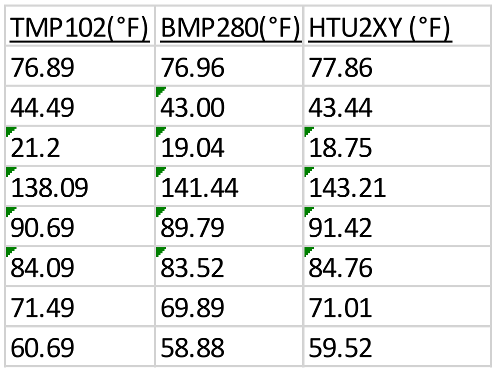


Figure 7.2: System Demonstration

Several temperature ranges were measured by the device, and the temperature results from each of the three sensor ICs were recorded. The sensor was placed into different temperature environments and given 15 to 30 minutes to acclimate. The results are summarized in Table 7.1.

Table 7.1: Temperature Measurement Results



TMP102(°F)	BMP280(°F)	HTU2XY (°F)
76.89	76.96	77.86
44.49	43.00	43.44
21.2	19.04	18.75
138.09	141.44	143.21
90.69	89.79	91.42
84.09	83.52	84.76
71.49	69.89	71.01
60.69	58.88	59.52

To measure the read range of the system, a testing setup which keeps the antenna of the smartphone perpendicular to the antenna of the sensor tag was required. To accomplish this, the smartphone was fixed onto an adjustable height platform which was placed on a table. The sensor tag was placed directly onto the table, resulting in an approximately parallel arrangement between the smartphone antenna and the sensor device antenna. The platform was slowly lowered down towards the sensor tag, until the smartphone could read data from the tag. At each height decrement, the tags position was adjusted on the table to make sure that the antennas were properly aligned. Due to the short lengths involved in reading the tag, the distances were measured using digital calipers.

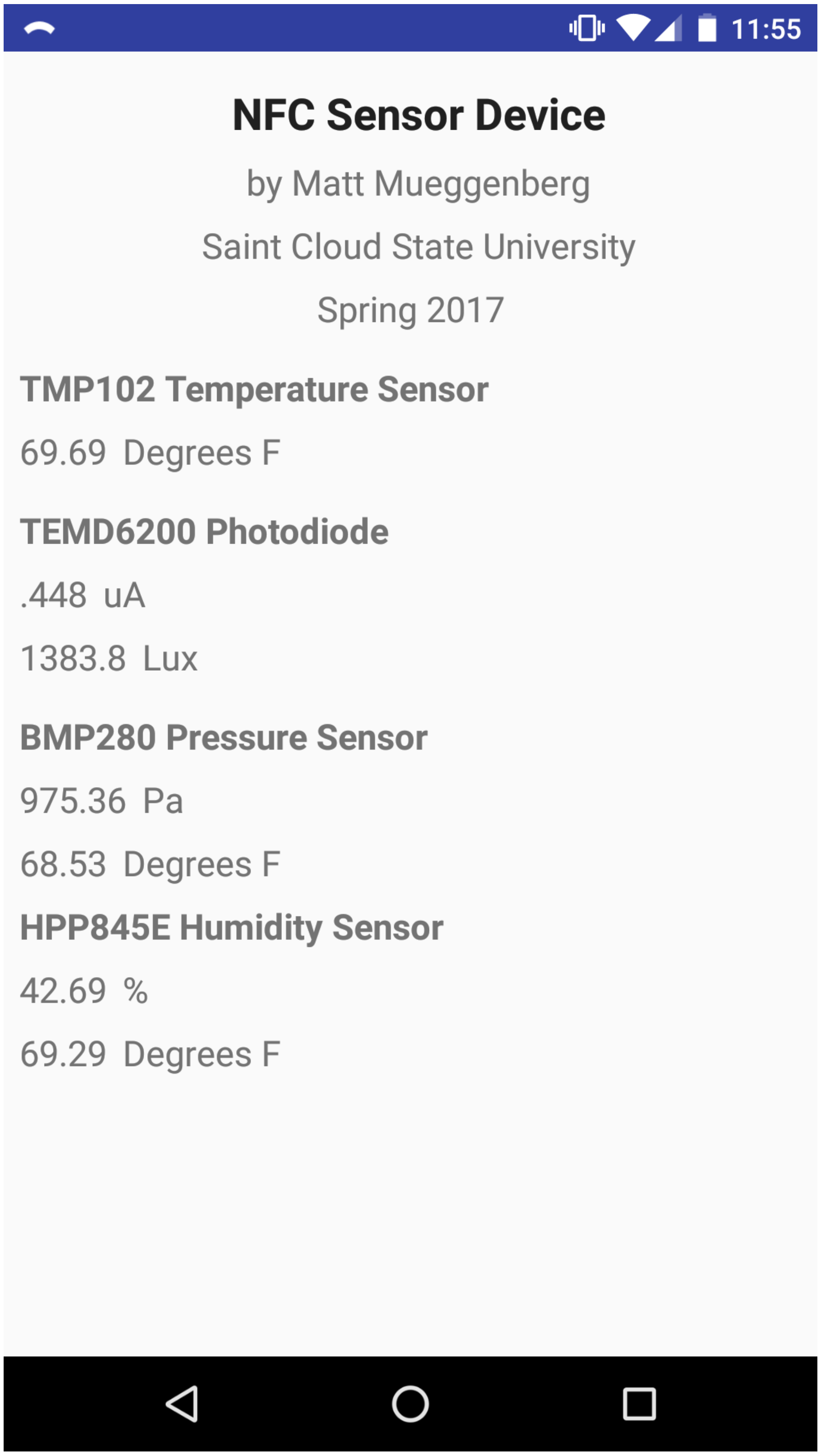
Three different distance thresholds were discovered. At the farthest distance, threshold 1, the tag was discovered through an intent on the smartphone application, but there was not enough power transferred to the tag for the microcontroller to run. This was indicated by the Android toast reporting that the tag was discovered, however the LED on the sensor device did not illuminate. At the next distance, the tag sent some data to the phone, but was often interrupted, or the data was not able to be read by the smartphone. This was indicated by the sensor tag LED turning blue when aligned with the smartphone antenna. The final threshold was at the closest distance. This was defined by repeatable successful reading of the sensor tag, indicated by the green LED illuminating. To get a consistent reading, threshold 3 should be used. This experiment was repeated three times, and the results are summarized in Table 7.2 below.

Table 7.2: Read Range Measurements

Threshold	**Test 1**	**Test 2**	**Test 3**
1	26.91mm	25.57mm	28.72mm
2	20.58mm	20.81mm	21.27mm
3	16.04mm	17.46mm	16.41mm

The read range of the sensor is only reliable when within 16mm of the smartphone antenna. This is caused by the inherently short range nature of NFC. It is short range by design, due to an inefficient power transfer. The sensor device antenna is much smaller than the smartphone antenna, which also dramatically reduces read range. Overall, the significance of the read range of an NFC device is small. NFC is designed to be short range, and in this case the system works well at 16mm or closer.

A screenshot of the device transaction is shown Figure 7.3. Qualitative verification of the photodiode was done by taking a lux meter and comparing the results at the same distance from a light source. The air pressure data is verified by comparing the pressure to typical levels at the current altitude. Finally, relative humidity is verified by checking weather information and comparing the sensor output to outside humidity levels.



11:55

NFC Sensor Device

by Matt Mueggenberg

Saint Cloud State University

Spring 2017

TMP102 Temperature Sensor

69.69 Degrees F

TEMD6200 Photodiode

.448 uA

1383.8 Lux

BMP280 Pressure Sensor

975.36 Pa

68.53 Degrees F

HPP845E Humidity Sensor

42.69 %

69.29 Degrees F

Figure 7.3: Transaction Example Screenshot

Chapter 8: Conclusion

NFC is a communications method with a specific niche application set. Advantages to NFC include security, wireless, and power transfer capabilities. NFC was the focal point of this star paper project. Research on the background of NFC was conducted and presented.

The intention of the project was to develop an NFC system which demonstrates the advantages of NFC. The central effort was a wireless, battery-less NFC enabled sensor device circuit which read sensor data and transmitted it to a smartphone using NFC. The system was designed and implemented on a PCB. Embedded firmware to control the sensor device was written, and a smartphone application for displaying the sensor data was designed.

Future improvements to the project could include providing for a method for sending data and displaying information in the opposite direction of this project, for example displaying data on the device instead of the smartphone. Other improvements could be made to the smartphone application, including local storage of the data, and continuous reading of the sensor data.

The project successfully demonstrated wireless transmission of sensor data from a battery-less device and displayed the data on the smartphone. The data was verified and the maximum read range was measured.

References

[1] Chang, H. (2015). *Everyday NFC: near field communication explained*. United States: Hsuan-hua Chang.

[2] Igoe, T. (2014). *Beginning NFC: near field communication with Arduino, Android, and PhoneGap*. Sebastopol: OReilly.

[3] Coskun, V., Ok, K., & Ozdenizci, B. (2012). *Near field communication from theory to practice*. Hoboken, NJ: Wiley.

[4] RF430FRL152H (ACTIVE). (n.d.). Retrieved February 30, 2017, from http://www.ti.com/product/RF430FRL152H?keyMatch=rf430frl&tisearch=Search-EN

[5] NXP Semiconductors. (2016, February 3). NT3H2111/NT3H2211. Retrieved from http://www.nxp.com/documents/data_sheet/NT3H2111_2211.pdf
Rev 3.0

[6] Microchip Technology Inc. (2015, December 8). PIC16(L)F1705/9. Retrieved from http://ww1.microchip.com/downloads/en/DeviceDoc/40001729C.pdf

[7] Texas Instruments. (2015, December). TMP102 Low-Power Digital Temperature Sensor with SMBus and Two-Wire Serial Interface in SOT563. Retrieved from http://www.ti.com/lit/ds/symlink/tmp102.pdf

[8] Bosch Sensortec. (2016, November 2). BMP280 Digital Pressure Sensor. Retrieved from https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMP280-DS001-18.pdf
Rev 1.18

[9] TE Connectivity. (2015, September). HTU20P(F) RH/T Sensor IC Miniature Relative Humidity and Temperature. Retrieved from http://www.te.com/commerce/DocumentDelivery/DDEController?Action=srchrtrv&DocNm=HPC203_4&DocType=Data Sheet&DocLang=English&DocFormat=pdf&PartCntxt=CAT-HSC0003

[10] Vishay Semiconductors. (2014, April). Ambient Light Sensor TEMD6200FX01. Retrieved from https://www.vishay.com/docs/81812/temd6200.pdf
Rev 1.5

[11] Elementary Transimpedance Photodiode Amplifier Design. (2012, August 27). Retrieved from http://www.jensign.com/PhotodiodeAmpDesign/

[12] STMicroelectronics. (2016, December). How to design a 13.56 MHz customized antenna for ST25 NFC / RFID Tags. Retrieved from 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 Rev 2

[13] Bevelacqua, P. (n.d.). NFC Antennas. Retrieved from http://www.antenna-theory.com/definitions/nfc-antenna.php

[14] OSH Park. (n.d.). Retrieved March 10, 2017, from https://oshpark.com/

[15] Learning the Physics Behind NFC. (n.d.). Retrieved May 15, 2017, from http://www.nfcdynamictag.com/antennadesign

[16] NFC Basics. (n.d.). Retrieved from https://developer.android.com/guide/topics/connectivity/nfc/nfc.html

[17] Reading NFC Tags with Android. (n.d.). Retrieved March 25, 2017, from https://code.tutsplus.com/tutorials/reading-nfc-tags-with-android--mobile-17278

[18] NfcA. (n.d.). Retrieved from https://developer.android.com/reference/android/nfc/tech/NfcA.html