Iowa State University

Senior Design May15-23

Final Report

Wireless Embedded Roadway Health Monitoring

Authors:
Mitchell Balke
Brandon Maier
Johnnie Weaver
Brandon Wachtel
Tyler Fish
Christofer Sheafe
Trieu Nguyen

Clients:
Dr. Halil Ceylan
Dept. of Civil, Construction, and Environmental Engineering

Advisors:
Dr. Daji Qiao
Dr. Jiming Song
Dr. Tie Qiu
Jeramie Vens

April 29, 2015

Contents

0 LIST OF ABBREVIATIONS AND ACRONYMS 3

1 EXECUTIVE SUMMARY 3

2 PURPOSE 3

3 DESIGN REQUIREMENTS 3

4 SUBSYSTEM REQUIREMENTS 4
4.1 COMMUNICATIONS . . . 4
4.1.1 SYSTEM REQUIREMENTS . . . 4
4.1.2 ANTENNA . . . 4
4.2 MICROCONTROLLER . . . 4
4.2.1 SYSTEM REQUIREMENTS . . . 4
4.2.2 FUNCTIONAL REQUIREMENTS . . . 5
4.2.3 NON-FUNCTIONAL REQUIREMENTS . . . 5
4.3 SENSOR TEMPERATURE/HUMIDITY . . . 5
4.3.1 SYSTEM REQUIREMENTS . . . 5
4.3.2 FUNCTIONAL REQUIREMENTS . . . 5
4.3.3 NON-FUNCTIONAL REQUIREMENTS . . . 5
4.4 POWER SUPPLY . . . 5
4.4.1 BATTERY . . . 5
4.4.2 POWER TRANSMISSION . . . 6
4.4.3 CHARGING CHIP . . . 7
4.5 BASE STATION . . . 7
4.5.1 SYSTEM REQUIREMENTS . . . 7
4.5.2 FUNCTIONAL REQUIREMENTS . . . 7
4.5.3 NON-FUNCTIONAL REQUIREMENTS . . . 7
4.5.4 DATA EXTRACTION . . . 7
4.6 ENCLOSURE . . . 7
4.6.1 FUNCTIONAL REQUIREMENTS . . . 7
4.6.2 NON-FUNCTIONAL REQUIREMENTS . . . 8

5 TESTING 8
5.1 COMMUNICATIONS . . . 8
5.1.1 AIR . . . 8
5.1.2 CONCRETE . . . 8
5.2 POWER . . . 8
5.2.1 BATTERY LIFE AND HEALTH . . . 8
5.3 DATA RETRIEVAL . . . 9
5.4 ENCLOSURE . . . 9

6 DETAILED DESCRIPTION 9
6.1 I/O . . . 10
6.2 INTERFACE . . . 10
6.3 HARDWARE . . . 10
6.3.1 MICROCONTROLLER . . . 10
6.3.2 RF TRANSCEIVER . . . 10
6.3.3 SENSOR . . . 10
6.3.4 FLASH MEMORY . . . 10
6.3.5 ENCLOSURE . . . 10

6.3.6 CHARGING CHIP . . . 12
6.4 SOFTWARE SPECIFICATIONS . . . 12
6.5 SCHEMATICS . . . 12
6.6 IMPLEMENTATION ISSUES AND CHALLENGES . . . 13

7 PARTS LIST **14**
7.1 PCB PARTS . . . 14
7.2 ENCLOSURE . . . 14
7.3 TRANSMITTER FOR CHARGING . . . 15

8 RESULTS **16**
8.1 COMMUNICATION . . . 16
8.2 CHARGING . . . 16
8.2.1 POWER TRANSMISSION . . . 16
8.2.2 CHARGING CHIP . . . 17
8.2.3 BATTERY LIFE . . . 18
8.3 ENCLOSURE . . . 18

9 CONCLUSION **19**

Appendices **20**

A OPERATIONAL MANUALS **20**
A.1 BASE STATION . . . 20
A.2 CHARGING . . . 20
A.3 ENCLOSURE . . . 20

B Alternative Project Plans/Designs **23**

C Other Considerations or Mistakes **24**
C.1 Charging Chip . . . 24
C.2 RF Charging . . . 24
C.3 Soldering . . . 24

D Team Contributions **25**

E CODE **26**
E.1 MAIN PCB CODING . . . 26
E.2 BASE STATION CODING . . . 55

References **66**

List of Figures **67**

0 LIST OF ABBREVIATIONS AND ACRONYMS

CAD Computer Aided Design

EM Electromagnetic

FCC Federal Communications Commission

FIFO First-In First-Out

I2C Inter-Integrated Circuit

ISM Industrial, Scientific, and

LCD Liquid Crystal Display

PCB Printed Circuit Board

RF Radio Frequency

RTC Real Time Clock

SD Secure Digital

SPI Serial Peripheral Interface

USB Universal Serial Bus Medical

ABS Acrylonitrile Butadiene Styrene

1 EXECUTIVE SUMMARY

The goal for this project was to design, develop, and implement a working system, which collects and monitors data that road surveyors normally complete. These tasks include sensor data sampling of concrete, charging remote systems wirelessly, and transmitting data across multiple nodes to a collection box on the side of the road. In addition, this system must be able to survive in concrete with minimal time intervals for charging.

2 PURPOSE

Structural health monitoring systems evaluate structures for safety without requiring the presence of an inspector. This saves time, money, and possibly lives as structural degradation can be detected much sooner than manual inspection. Implementing such a system without wireless communication becomes too difficult, fragile, and expensive to be feasible. A wireless sensor network makes the system low cost, have quick installation times, and high system reliability.

3 DESIGN REQUIREMENTS

The new systems designed by this project as well as the improvements to the existing systems will all be expected to meet the following criteria.

- Sensor can communicate between multiple nodes.
- The enclosure is water/shock resistant and can handle pressures induced by the solidification of concrete and overhead traffic.
- Handles temperature ranges from -20°F to 140°F.
- The battery life of each unit will last a minimum of one year.
- Each charging of the battery will take a maximum of 12 hours.
- Must be able to transmit and receive data between nodes through concrete.
- Must encompass full automation of data aggregation, transmission, & receiving.
- Must be able to detect and re-route around non-functional nodes.
- The base station must store all data logs
- Log files must include date, time, nodes used, and information about the samples.

4 SUBSYSTEM REQUIREMENTS

The wireless node network is composed of several subsystems with different responsibilities. Subsystems in the node include communications, a micro-controller for processing, sensors, power transmitting circuits, and a centralized base station to collect data. Communication is conducted wirelessy through the concrete via RF transceivers. The micro-controller is tasked with scheduling measurements, controlling the flow of communication between nodes and the base station, and controlling power states to conserve energy. Sensors installed are responsible for measuring temperature and relative humidity.See Figure 1 for an overview of the system.

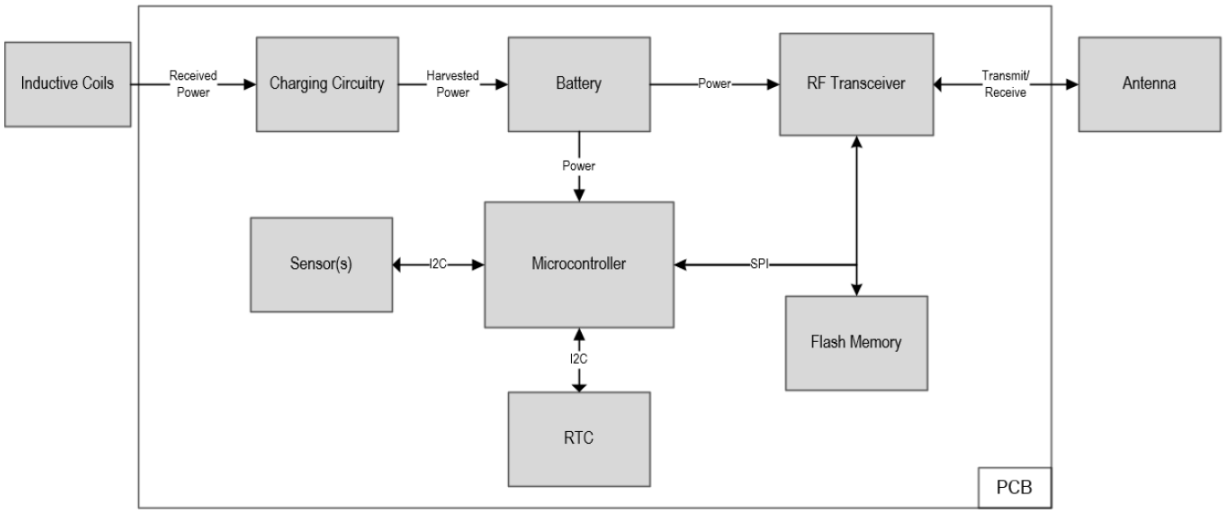

Figure 1: System block diagram

4.1 COMMUNICATIONS

4.1.1 SYSTEM REQUIREMENTS

Communication between the nodes will be accomplished with an RF transceiver operating in the 433MHz ISM band. Communication software must be designed for maximum reliability to prevent packet loss which requires re-transmission. In addition, components for the communication circuit are selected to minimize current draw as well as the size of the circuit.

4.1.2 ANTENNA

The antenna used for communication is an enclosed whip antenna operating at 433MHz. The antenna meets the requirements of having sufficient reception in concrete, durability and low cost.

4.2 MICROCONTROLLER

4.2.1 SYSTEM REQUIREMENTS

The node is controlled by a low-power micro-controller augmented with a real-time clock. The micro-controller communicates with the RF transceiver and the flash memory device via an SPI bus and with the real-time clock via an I2C bus. Communication with the temperature and humidity sensor is done using Sensirion's proprietary serial communication protocol.

4.2.2 FUNCTIONAL REQUIREMENTS

The micro-controller selected and all software running on it must be optimized for the lowest possible power consumption to maximize battery life.

4.2.3 NON-FUNCTIONAL REQUIREMENTS

Once deployed, it will be impossible for the user to reprogram or reset the device, meaning system failure is unacceptable. Device failure will result in data loss and will increase the power consumption of nearby nodes affecting the overall lifespan of the network. The primary solution is designing the device as reliable as possible with fault-tolerant software and performing energy-efficient optimization to system and network functions in order to expand system life expectancy.

4.3 SENSOR TEMPERATURE/HUMIDITY

4.3.1 SYSTEM REQUIREMENTS

The micro-controller will interface with a Sensirion SHT10 temperature and humidity sensor to make the required measurements.

4.3.2 FUNCTIONAL REQUIREMENTS

Sensor measurements must be taken in such a way as to minimize power consumption and not interfere with other peripherals on the device. Communication with the sensor is done with a non-standard protocol, therefore, a hardware solution does not exist. Sensor measurements can take up to 250ms so the software solution must be designed to avoid performing this task while any other time-sensitive requirements exist to the micro-controller.

4.3.3 NON-FUNCTIONAL REQUIREMENTS

There is no benefit to leaving the sensor in sleep mode while measurements are not being taken, so the sensor will not be powered directly by the battery but rather by a general purpose pin on the micro-controller, allowing the sensor to only be powered when a measurement needs to be taken. The software for sensor communication must allow the correct amount of time for the sensor to initialize before starting any communication.

4.4 POWER SUPPLY

The power source must charge the nodes wirelessly. This is due to the harsh conditions during the curing process and setup of concrete that causes a high failure rate of wired connections. For the devices to be wireless, it must be powered by an internal battery.

4.4.1 BATTERY

The battery chosen will use Lithium Ion technology due to Lithium Ion being able to function at temperatures ranging from -20°F to 140°F. The battery must also be able to last at least one half year before needing recharging. Due to the enclosure's size constraints the battery package size needs to be small. The current battery selected meets the requirements for both size and battery life. Estimations for the batteries life span to be six to seven years of consumable power without the need of charging. Calculations for the life span of the battery have it lasting between six and seven years.

4.4.2 POWER TRANSMISSION

The node will use a method of wireless power transmission. Multiple methods can be used to charge the circuit like RF, Thermo-electric, or Piezoelectric. However, the method that resulted in the best efficiency through concrete was magnetic resonant coupling. Specifically, loosely magnetic coupling was chosen for this project. With strong magnetic coupling, the nodes would require more length to the enclosure that is outside of the size constraints of the project. This resulted in picking loose magnetic coupling since the coil is able to fit inside the node and not strain the size requirements. The size of the receiving coil is bounded by the node's housing (5cm x 5cm).

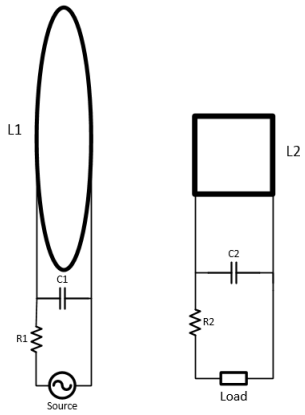

Figure 2: Wireless Power Transmission

The transmitter was created to resemble a Royer Oscillator circuit with an LC tank consisting of a copper coil and a few capacitors. The transmitter is driven by a DC power source with the source voltage limited between 12V and 20V. Higher voltage will generate higher power at the coil that introduces an AC voltage across the LC tank. The LC tank defines the oscillating frequency, which is approximately 2.189 MHz. The transmitter's circuit can be seen in Figure 3.

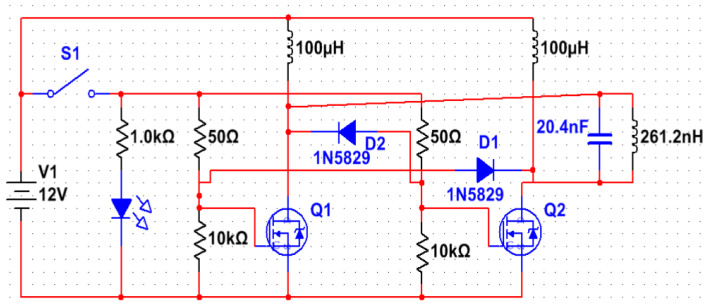

Figure 3: Transmitter - Royer Oscillator Circuit

The receiver consists of a copper coil and a capacitor that is tuned to the same resonant frequency of 2.189 MHz. The receiving coil will be a square to maximize the area and to meet the size requirements of the enclosure. The same material as the transmitters coil was used. See Figure 4.

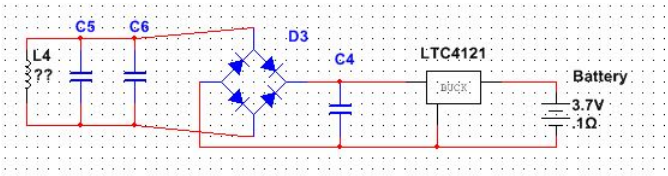

Figure 4: Receiver

4.4.3 CHARGING CHIP

The charging chip must be able to charge the Lithium Ion battery to meet the 12 hour charging requirement. The chip can be configured to disconnect the battery during low battery voltage conditions, but currently this feature is not being used. This chip was chosen for being up to 95% efficient which is crucial for meeting the 12 hour charging time requirement.

4.5 BASE STATION

4.5.1 SYSTEM REQUIREMENTS

The base station is a centralized subsystem of the network that receives information from the installed nodes as well as serving as the location for data extraction.

4.5.2 FUNCTIONAL REQUIREMENTS

The base station has two duties: to initialize the network links upon installation and to receive and capture data collected by nodes. Nodes are displaced from their initial locations during the concrete mixing process. The base station detects how far away nodes are by sending power at different levels. Nodes are then assigned "depth" levels based upon what power levels the base station is sending.

4.5.3 NON-FUNCTIONAL REQUIREMENTS

The base station is placed on the side of the road to be easily accessible to road workers. Since the base station is not enclosed in the road, available power requirements are not defined. Additionally, the power of the nodes can be configured through the base station. The base station can tell the nodes to transmit with less power to conserve energy or to boost the signal strength for better communication. The base station should also be able to change the frequency at which nodes take and send data. Additionally, the sample rate is configurable from the base station to control the amount of time the nodes are active.

4.5.4 DATA EXTRACTION

The data is collected when the base station correctly receives a transmission sent by a node. The raw data is stored within a microSD card which can be removed and interpreted at a work computer by a road worker. For a more refined system the base station would relay information via cellular network.

4.6 ENCLOSURE

4.6.1 FUNCTIONAL REQUIREMENTS

Due to the nodes being placed in concrete the enclosure must be able to protect the internal circuitry from the negative effects of the surrounding environment. The enclosure must be watertight to keep water and humidity from the concrete mixing and pouring process from reaching the electronics. It must also be able to protect the internal components from crushing pressures, around 30 psi, experienced during the paving process. During the curing process, the concrete environment is very acidic and the enclosure must be able

to resist the eroding nature of these acids. Finally, the enclosure must be able to withstand temperatures ranging from -20°F to 140°F that the road would experience throughout the year in Iowa.

4.6.2 NON-FUNCTIONAL REQUIREMENTS

The entire node will be put into a concrete roadway, therefore, the enclosure must be large enough that all of the internal electronics can fit within it but small enough not to cause structural integrity loss in the portion of the road where it is embedded. The end goal for future products would be to have a node that is on the scale of a piece of aggregate, around a half inch, allowing for the nodes to be tossed into cement trucks during the mixing process. The cost of the enclosure must also be reasonable, in that it shall not drive the node cost up to levels where large quantities of nodes being purchased would result in an unreasonable amount of money being spent.

5 TESTING

Testing was subdivided into two sections: the efficiency of wireless charging and the successful transmission and reception of data between nodes in the network. Wireless charging was tested by observing transmitted power through concrete slabs. Communication was tested by monitoring the success rate of characters transmitted over a large distance while the nodes were encased in concrete.

5.1 COMMUNICATIONS

Initial testing of communications was done by having two nodes talk to one another. The testing was done first by transmitting a string of characters through air and checking if the information was correctly received. Afterwards, the nodes were enclosed in concrete to observe if communication through concrete would work.

5.1.1 AIR

Two nodes were separated by a distance of a few feet in open air. One node was programmed to transmit continuously with the other set to receive data and check for errors. The goal of this stage was to check that success rate of communication and that error checking was functioning properly.

5.1.2 CONCRETE

Concrete testing was performed in two stages. First, two nodes were surrounded in concrete blocks and tested in the same way as testing through air was performed. Second, a node was buried in a hole drilled into a parking lot. This node was programmed to wake from sleep mode and test communication every five minutes. When the node entered active mode, it sent the same packet five times at different output powers, -5dBm, 0dBm, +5dBm, +7dBm, and +10dBm. The receiver first sat on the surface of the concrete, and then in another hole 4 feet away to test longer distance communication through concrete.

5.2 POWER

For testing the power consumption of a node, current draw, the average lifespan of the battery under normal operation, and the efficiency of the charging system will be measured. This will allow for calculations of node and battery lifespan.

5.2.1 BATTERY LIFE AND HEALTH

To test the battery, measurements will be taken of the remaining battery life, in voltage, after 24 hours of operation. This will give the estimated battery life and whether or not the size of the battery being used should be increased. Ideally, measurements will be taken to see how exposing the battery to the two extremes in temperature ranging from -20°F to 140°F affects the lifespan of the battery.

5.3 DATA RETRIEVAL

The sensor in the device was verified for accuracy. It was exposed to the elements to test if it can survive pouring while providing accurate temperature and humidity readings. The accuracy can be tested by placing it in a controlled environment and measuring the accuracy across different ranges of humidity and temperature. A parking lot was chosen as the testing environment since nodes could be recovered with little hassle while still emulating the expected conditions found in a roadway. Finally, verification that the sensor can survive concrete pouring will be needed. The best way to perform this test is by subjecting it to high temperature and acidity levels similar to those present in the curing process of concrete.

5.4 ENCLOSURE

The purpose of the enclosure is to protect the electronics contained within from the concrete environment in which the system will be embedded. The enclosure must be able to protect the equipment from water and acids within the concrete during the curing process, withstand the pressures induced while paving, and endure the large temperature swings experienced throughout the year. ABS plastic was used as the material for the enclosure due to its strength and resistance to acids.

Testing of the enclosure will be subject to a waterproofing test as well as a strength test. To test that the enclosure is waterproof, a full enclosure will be built and submerged in concrete. After the concrete has hardened, the enclosure will be dug up and opened to verify that no water has made it into the inside of the enclosure. The strength test will replicate the pressures that the enclosure will experience while the road is being paved. The enclosure will be set on a flat surface and weights will be added on top until the desired weight and pressure are created. The enclosure will pass the test if no structural damage is seen after removing the weights. See Figure 5.

Figure 5: Node encased in concrete after recovery

6 DETAILED DESCRIPTION

This section will lay out the main specifications set forth for the project by the team and advisers. It will include sections on I/O specifications, interface specifications, hardware and software specifications, simulations, implementation issues, and testing.

6.1 I/O

The base station will serve as the only point of I/O for the user. Output generated by the base station will be a feed from the network containing sensor data, the data can be stored in a file or outputted to stdio for live viewing. There will be no continuous user input into the system, and the only time input to the system is needed is when the user wants to change network parameters such as sampling frequency.

6.2 INTERFACE

The base station is a raspberry Pi running Raspbian (Linux) without a graphical user interface. It can be interfaced with any traditional method of accessing Linux terminals. This includes connecting an external HDMI display with keyboard, attaching a WiFi dongle, and connecting via SSH. When deployed for data collection, data gathered from the network can be stored onto the raspberry's SD card or a USB flash drive.

6.3 HARDWARE

6.3.1 MICROCONTROLLER

The choice of micro-controller is the Texas Instruments MSP430F2122. This device is ideal for our application due to its low power consumption, application flexibility, and large amount of RAM. The MSP430 has an active power consumption of $250\mu A$, standby of $0.7\mu A$, and 4KB of flash memory. The MSP430 supports many features that are important to the project like I2C, SPI, configurable clocks, and a large supply of GPIO pins (24).

6.3.2 RF TRANSCEIVER

We decided upon the use of the Texas Instruments CC1101 for wireless communication. The device operates in the ISM 433MHz frequency band. This component was selected for relatively low-power consumption, flexibility, and minimal required external passive components. The benefit of operating in an ISM band is that a license is unneeded for transmissions with power less than a watt. Additionally, operating at 433MHz offers the least amount of signal and power attenuation while being the lowest frequency device commercially available. A simple whip-antenna designed for 433MHz communication was chosen based on price, size, and relatively good performance.

6.3.3 SENSOR

The Sensirion's SHT10 is used for the temperature and humidity measurements. The device offers accurate readings and a package that suits our enclosure and is in an acceptable price range. The SHT10 is accurate to within 4.5RH and 0.5°C, has a footprint of 7.5x5mm, and costs $7. Higher accuracy sensors are available, but cost in excess of $25.

6.3.4 FLASH MEMORY

The Microchip SST25VF040B 4Mb serial flash memory chip is used to store local sensor data before being sent to the base station.This device was chosen for its low cost and SPI-compatibility. To conserve power, this chip was chosen as flash memory is non-volatile, so it can be powered down when not being accessed. The chip is powered by a pin on the micro-controller rather than the battery so that it can be powered off.

6.3.5 ENCLOSURE

The enclosure was 3-D printed using ABS plastic. The material protects the circuitry from the concrete, but does not interfere with the wireless communications or inductive charging. Plastic was chosen because it contains no conductive materials that would interfere with communication and charging transmission.

There are two different designs for the enclosure, one for charging and one for non-charging. The top and bottom parts of the enclosure have a thickness of 0.2 inches and when the two parts are put together, the thickness of the four walls is the same. The sizes and designs for both inner and outer parts of the enclosure, for both charging and non-charging designs, can be seen in Figure 6, Figure 7, and Figure 8.

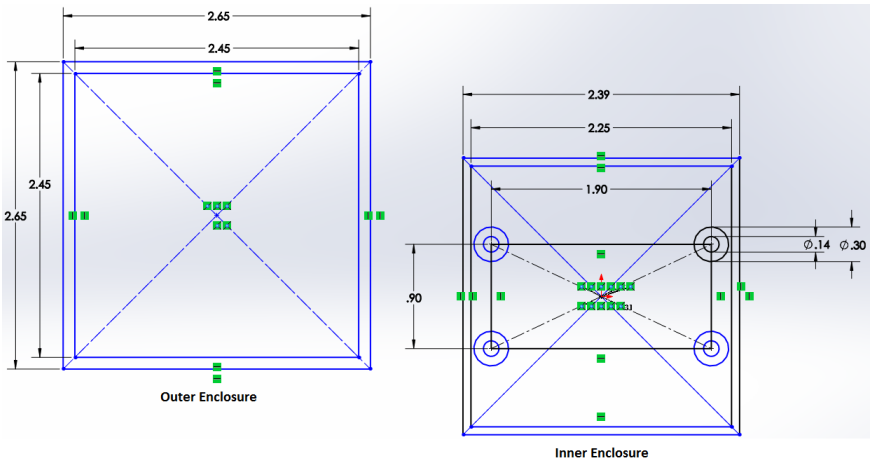

Figure 6: Top view of design for both charging and non-charging enclosures

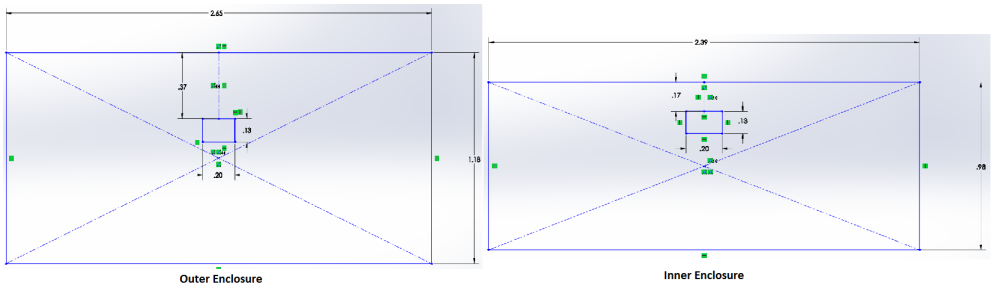

Figure 7: Front view of design for non-charging enclosure

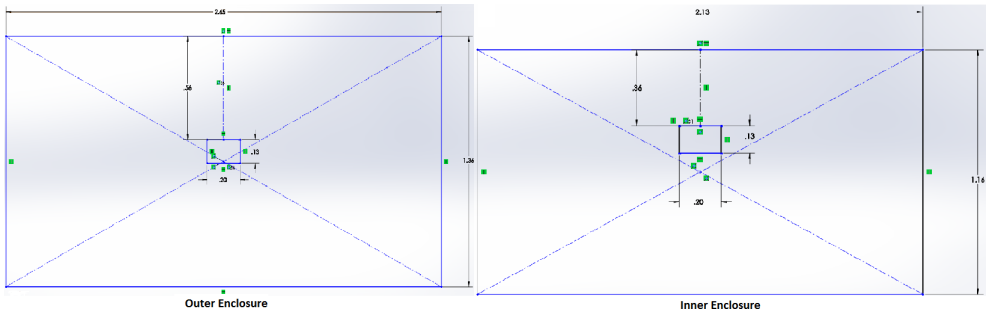

Figure 8: Front view of design for charging enclosure

6.3.6 CHARGING CHIP

The chip used for charging the Lithium Ion is Linear Technologies LTC4121. This chip was chosen for its high efficiency. The chip is up to 95% efficient. The chip also has a wide input voltage range of 4.4V to 40V which is needed when charging with inductive coils. If the coils get too close together, the voltage will rise and the charging chip must be able to handle this high voltage without burning up.

The chip is also ideal for being configurable. The float voltage of the battery was configured to be 3.7 volts. This was to protect the micro-controller, antenna, and other circuitry from higher voltages than 3.7 volts. See Figure 9 for schematic.

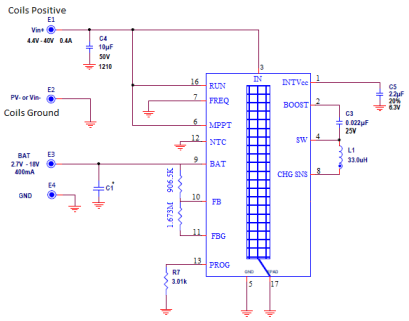

Figure 9: Charging Chip Circuitry

6.4 SOFTWARE SPECIFICATIONS

Software is optimized for lowest possible power consumption on battery powered devices. This is to maximize lifetime, as the node needs to be able to survive as long as possible on one charge. This is achieved by minimizing the time spent with the device awake. For instance no complex calculations are done by the microcontroller e.g. floating-point or math.h functions. In order to further decrease active run-time, static variables are preferred over passing function values to decrease function switching time and allow the compiler to further optimize the code.

Similarly, the software is optimized to have a small memory size. The MSP430 only has 512B of RAM available, so no large data-structures are used in the software. To allow for the storage of sensor data, a secondary flash memory is used.

6.5 SCHEMATICS

The hardware schematics for the Royer oscillator based transmitter and the receiving system can be seen in Figure 3 and Figure 4 respectively in section 4.4.3.

The PCB schematics were made in NI Ultiboard. Figure 10 shows the schematic for the non-charging PCB while Figure 11 shows the charging PCB.

Using Multisim, the PCBs were created for both the charging and non-charging variations. Figure 12 shows the Multisim diagram for the charging PCB.

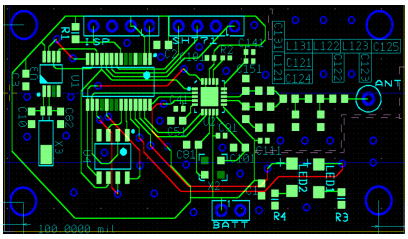

Figure 10: Non-Charging PCB

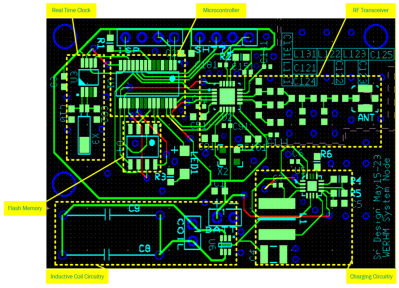

Figure 11: Charging PCB

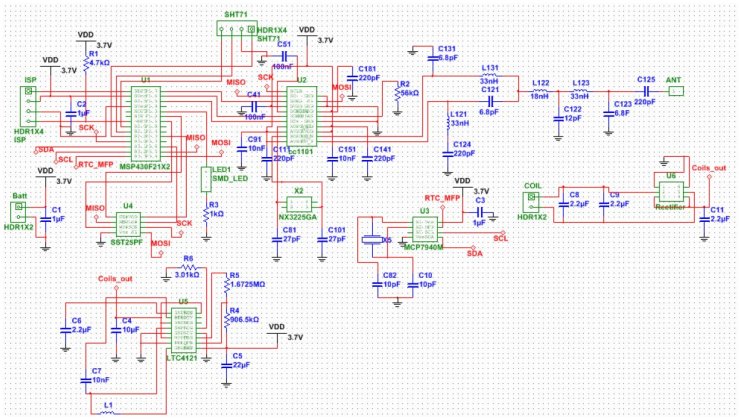

Figure 12: Multisim schematic for the charging PCB.

6.6 IMPLEMENTATION ISSUES AND CHALLENGES

One main challenge of implementation for the nodes is survivability during the concrete pouring and curing. During the concrete pouring, there will be high temperatures and pressures as well as strong vibrations which can cause stress to the PCB and enclosure. After being installed, the acidity of the concrete can corrode the electronics and effectively destroy the node. The enclosure was designed specifically to addressed these issues.

Signal attenuation in the concrete was another challenge for the system. The signal loss increases drastically in concrete compared to air, which may cause the system to transmit data repeatedly until success. This will cause the batteries to drain faster than desired. The nodes are designed to transmit data and wait for a data received signal. If the node does not receive the signal, it will re-transmit at a higher output power and repeat the process until transmission is received.

7 PARTS LIST

7.1 PCB PARTS

For this project, there were two separate designs which require to different parts lists. The two designs consisted of a non-charging pcb and a charging pcb. These two circuits only differ by the addition of the charging chip, receiving coil, and the additional passive components required for charging. The best representation of the two different designs can be seen in Figure 10 and Figure 11. Figure 13 shows each component and pricing.

Label	Value	Qty	Manufacturer	Mfctr. Part #	Description	Cost / part	Subtotal
BATT		1	Ultralife	UBP002	Battery Packs 3.7V 950mAH LITH REC MAX LEAD LENGTH 2.5"	$10.01	$10.01
C1, C2, C3	0.01uF	3	Murata Electronics	GCM188R71H103KA37D	Multilayer Ceramic Capacitors MLCC - SMD/SMT 0603 0.01uF 50volts X7R 10%	$0.10	$0.30
C10, C82	7pF	2	TDK	C1608C0G1H070D080AA	Multilayer Ceramic Capacitors MLCC - SMD/SMT 0603 50volts 7.0pF C0G 0.5pF	$0.10	$0.20
C111, C151, C181	220pF	3	Murata Electronics	GRM1555C1H221JA01D	Multilayer Ceramic Capacitors MLCC - SMD/SMT 0402 220pF 50volts C0G 5%	$0.10	$0.30
C122	8.2pF	1	Murata Electronics	GCM1885C1H8R2DA16D	Multilayer Ceramic Capacitors MLCC - SMD/SMT 0603 8.2pF 50volts C0G +/-0.5pF	$0.11	$0.11
C123	5.6pF	1	Murata Electronics	GCM1885C1H5R6DA16D	Multilayer Ceramic Capacitors MLCC - SMD/SMT 0603 5.6pF 50volts C0G +/-0.5pF	$0.11	$0.11
C124, C125	220pF	2	Murata Electronics	GCM1885C1H221JA16D	Multilayer Ceramic Capacitors MLCC - SMD/SMT 0603 220pF 50volts C0G +/-5%	$0.11	$0.22
C131	3.9pF	2	Vishay / Vitramon	VJ0603A3R9BXQCW1BC	Multilayer Ceramic Capacitors MLCC - SMD/SMT 0603 3.9pF 10volts C0G +/-0.1pF	$0.08	$0.16
C41	100nF	1	Murata Electronics	GRM155R71C104KA88D	Multilayer Ceramic Capacitors MLCC - SMD/SMT 0402 0.1uF 16volts X7R 10%	$0.10	$0.10
C4*	10uF	1	Taiyo Yuden	UMK325BJ106KM-T	Multilayer Ceramic Capacitors MLCC - SMD/SMT X5R 1210 50V 10uF 10%	$0.74	$0.74
C5*	22uF	1	Murata Electronics	GRM188R60J226MEA0D	Multilayer Ceramic Capacitors MLCC - SMD/SMT 0603 22uF 6.3volts X5R + - 20%	$0.24	$0.24
C51	100nF	1	Murata Electronics	GCM188R71H104KA57D	Multilayer Ceramic Capacitors MLCC - SMD/SMT 0603 0.1uF 50volts X7R 10%	$0.13	$0.13
C6*	2.2uF	1	Murata Electronics	GRM155R60J225ME15D	Multilayer Ceramic Capacitors MLCC - SMD/SMT 2.2uF 6.3Volts 20%	$0.25	$0.25
C7*	0.022uF	1	TDK	C1005X7R1E223K050BB	Multilayer Ceramic Capacitors MLCC - SMD/SMT 0402 0.022uF 25volts X7R 10%	$0.10	$0.10
C8*	4700pF	1	WIMA	FKP1R014704D00JSSD	Film Capacitors 1250V 4700pF 5% PCM 15	$0.95	$0.95
C81, C101	27pF	2	Murata Electronics	GCM1885C1H270JA16D	Multilayer Ceramic Capacitors MLCC - SMD/SMT 0603 27pF 50volts C0G 5%	$0.10	$0.20
C9*	0.022uF	1	WIMA	MKP1O122204C00KSSD	Film Capacitors 1KV .022uF 10%	$1.16	$1.16
C91, C141	10nF	2	Murata Electronics	GRM155R71H103KA88D	Multilayer Ceramic Capacitors MLCC - SMD/SMT 0402 0.01uF 50volts X7R 10%	$0.10	$0.20
L1*	33uH	1	Bourns	SRN6028-330M	Fixed Inductors 33uH 20% SMD 6028	$0.40	$0.40
L121, L123, L131	27nH	3	Murata Electronics	LQG18HN27NJ00D	Fixed Inductors 27 NH 5%	$0.21	$0.63
L122	22nH	1	Murata Electronics	LQG18HN22NJ00D	Fixed Inductors 0603 22nH 5%	$0.20	$0.20
LED1		1	Wurth Electronics Inc	150141BS73100	WL-SMTW SMD TOP LED WATERCLEAR	$0.28	$0.28
LED2		1	Wurth Electronics Inc	150141AS73100	WL-SMTW SMD TOP LED WATERCLEAR	$0.28	$0.28
R1	47k	1	Vishay / Dale	CRCW060347K0FKEA	Thick Film Resistors 1/10watt 47Kohms 1%	$0.08	$0.08
R2	56k	1	Vishay / Dale	CRCW060356K0FKEA	Thick Film Resistors 1/10watt 56Kohms 1%	$0.08	$0.08
R3	1k	2	Vishay / Dale	CRCW06031K00FKEA	Thick Film Resistors 1/10watt 1.0Kohms 1%	$0.08	$0.16
R4*	909k	1	Vishay / Dale	CRCW0603909KFKEA	Thick Film Resistors 1/10watt 909Kohms 1%	$0.08	$0.08
R5*	1.68M	1	Vishay / Dale	CRCW06031M69FKEA	Thick Film Resistors 1/10watt 1.69Mohms 1%	$0.08	$0.08
R6*	3k	1	Vishay / Dale	CRCW06033K00JNEA	Thick Film Resistors 1/10watt 3.0Kohms 5%	$0.08	$0.08
U1		1	Texas Instruments	MSP430F2122IPWR	16-bit Microcontrollers - MCU 16B Ultra-Lo-Pwr Microcntrlr	$3.52	$3.52
U2		1	Texas Instruments	CC1101RGPR	RF Transceiver Low-Power Sub-1GHz	$4.62	$4.62
U3		1	Microchip Technology	MCP7940M-I/MS	Real Time Clock I2C RTCC 64B SRAM	$0.74	$0.74
U4		1	Microchip Technology	SST25VF040B-50-4C-SAF	Flash FLASH MEMORY 4M (512Kx8) 50MHz	$0.96	$0.96
U6*		1	Avago Technologies	HSMS-282P-TR1G	Schottky Diodes & Rectifiers 15 VBR 1 pF	$1.65	$1.65
X3		1	ABRACON	AB26TRQ-32.768kHz-T	Crystals 32.768kHz +/-20ppm 12.5pF -40C +85C	$0.45	$0.45
ANT		1	Anaren	66089-0406	Antennas Antenna For Module 04C	$1.94	$1.94
ANT		1	Amphenol RF	A-1JB	RF Connectors / Coaxial Connectors AMC PCB JACK S/M T&R U.FL Compatible	$0.69	$0.69
X2		1	NDK	NX3225GA-26MHZ-TI	CRYSTAL 26MHZ 10PF SMD	$0.79	$0.79
U5*		1	Linear Technology	LTC4121EUD#PBF	IC BATT CHARGER 40V 400MA 16QFN	$5.53	$5.53
							Total
							$33.19

Figure 13: Parts List

7.2 ENCLOSURE

The following parts lists cover the current designs of the enclosure for both the charging and non-charging models. The primary expense is the cost for 3-D printing both the inner and outer parts of the enclosure. The design for our system needed to be small, therefore we needed to have an enclosure that would perfectly fit the electronics that we were putting into the concrete. To do this, we were unable to find and purchase a simple case to modify to our needs and thus had to design and 3-D print the enclosures. This uniqueness required for the enclosure is what caused the price for the enclosure to go up. We purchased a GoreTex repair kit to use as the mesh for our design. The cost of the kit was low and many meshes can be made from a single kit, so an approximation of the total cost based on the percentage of the material used was made for the cost of the mesh materials. This approximation can be seen in the two enclosure parts lists. Epoxy was also purchased to hold the enclosure pieces together and again, only a small amount is needed to create

a single enclosure and multiple enclosures can be made from a single tube of epoxy. An approximation on the cost of the epoxy per node is made in the parts lists. See Figure 14 and Figure 15.

Part	QTY	Prices	Total Price
Non-Charging Outer Enclosure	1	$14.00	$14.00
Non-Charging Inner Enclosure	1	$18.00	$18.00
#6-3/8in Machine Screw	4	$0.04	$0.16
Epoxy	NA	$4.00	$0.25
Epoxy and Mesh Materials	NA	$4.00	$0.25
		Total	**$32.66**

Figure 14: Parts List for Non-Charging Enclosure

Part	QTY	Prices	Total Price
Charging Outer Enclosure	1	$16.50	$16.50
Charging Inner Enclosure	1	$20.00	$20.00
#6-3/8in Machine Screw	3	$0.04	$0.12
Epoxy	NA	$4.00	$0.30
Mesh Materials	NA	$4.00	$0.25
		Total	**$37.17**

Figure 15: Parts List for Charging Enclosure

7.3 TRANSMITTER FOR CHARGING

The following parts list covers the current design of the power transmitter circuit. Most of the passive components were rated for a higher current draw in case of higher power transmission being required. This will cause the circuit to be a bit more expensive. However some of the passive components could be cheaper if the current design was modified. The circuit only requires a few amperes ranging from 200mA to 1A while some of the component ratings are set for 3A to 5A. Additionally the relay can be swapped out for an ordinary switch as long as it's acceptable for the appropriate ratings.

The transmitter cost is not included in the total cost since charging is a separate design of the current project, which is wireless communication through concrete. Figure 16 shows the individual components along with each individual prices and total cost.

Part	QTY	Size	Part #	Manufacture	Mouser #	Manuf. Part #	Prices	Total Price
Diodes	2		1N4148	Fairchild Smiconductor	512-FYA3010DNTU	FYA3010DNTU	$ 1.68	$ 3.36
Mosfet	2		IRFZ44N	NXP Semiconductors	771-PSMN015-60PS127	PSMN015-60PS,127	$ 1.02	$ 2.04
Relay	1		G6A-234P	Omron electronics	653-G6A-234P-DC12	G6A-234P-ST-US-DC12	$ 6.08	$ 6.08
Inductor	2	100uH		Hammond Manufacturing	546-1539M05	1539M05	$ 6.62	$ 13.24
Resistor	2	50 Omh 5W	Supplied by ISU parts shop					$ -
	1	1K Omh 1/4W	Supplied by ISU parts shop					$ -
	2	10k Omh 1/4W	Supplied by ISU parts shop					$ -
Capacitors	3	6.8nF	WIMA FKP1	WIMA	505-FKP16800/630/5	FKP1J016804C00JSSD	$ 0.64	$ 1.92
Copper Tubing	14.8 inches	1/4 inch	Sold in rolls at hardware stores	25 ft roll /$16.99 per roll			$16.99	$ 0.84
							TOTAL	**$ 27.48**

Figure 16: Transmitter Part List and Cost

8 RESULTS

8.1 COMMUNICATION

Testing different output powers for communications testing through concrete showed packet success rates that greatly decreased as output power was lowered. Still, as shown in Figure 17,the output power was found to be +5dBm or higher, packet success rate is high enough that total data loss should be rare in a flooding routing scheme.

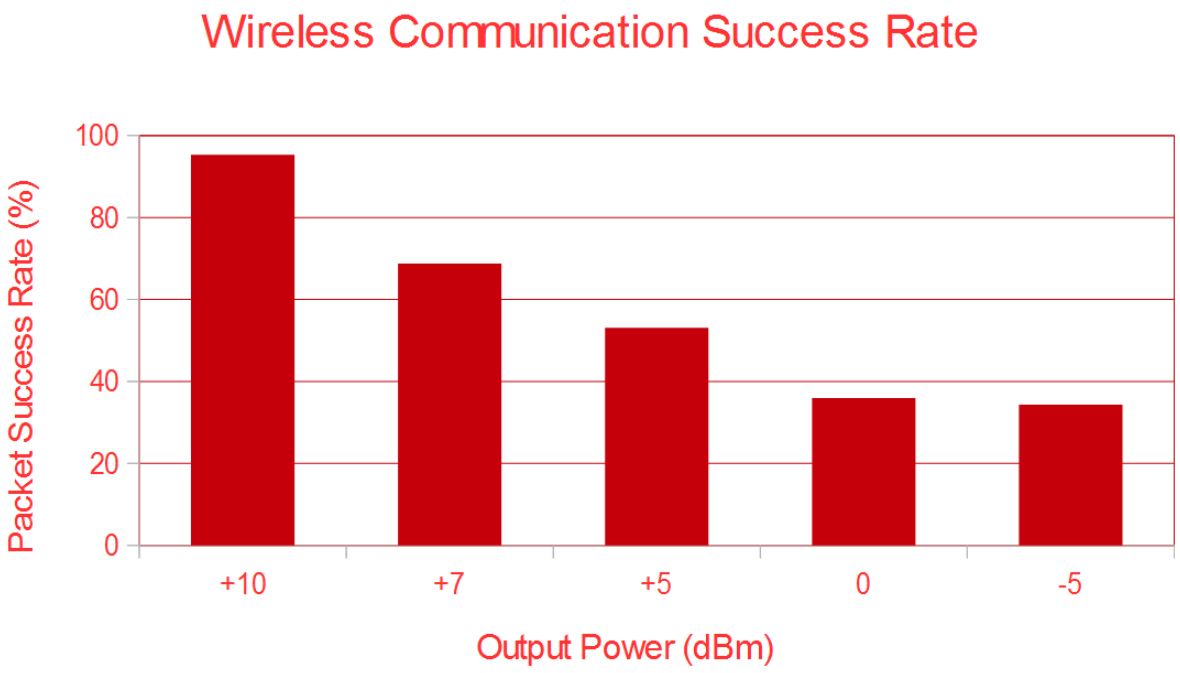

Figure 17: Packet success rate for different output powers

8.2 CHARGING

8.2.1 POWER TRANSMISSION

The receiver circuit needs to have a matching resonant frequency of 2.189 MHz. This is done with using the following equation.

$$Freq = \frac{1}{2 * \pi * \sqrt{L * C}}$$

Where L is the total inductance seen from the coil and C is the total capacitance. It is easier to find C after the coil is made. Currently the coil requires 26.7nF to get 2.189 MHz.

A load analysis was done to determine the best resistive load that should be applied to the charging circuitry. The ideal load was found to be about 100Ω as shown in Figure 18. With this load, the maximum power received was approximately 23 mW at a distance of four inches.

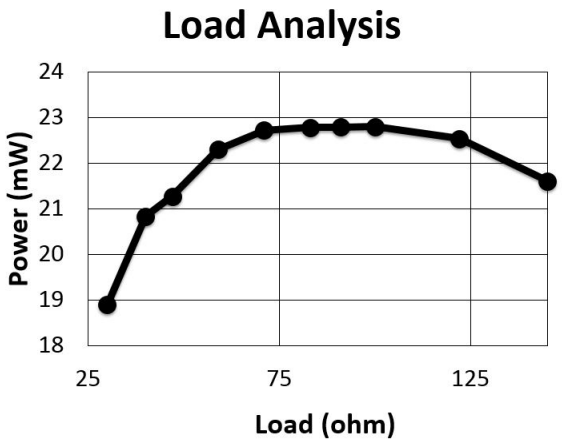

Figure 18: Receiver load analysis at four inches

Since loosely coupled magnetic resonant was implemented, the efficiency decreases drastically as distance increases as shown in the Figure 19. The efficiency was defined as power receiving at 100Ω divided by the power output at the source. The test was carried out with 12V power supply. Current at the source was measured to determine the transmitting power. Voltage across the load was measured to determine the power at the receiver.

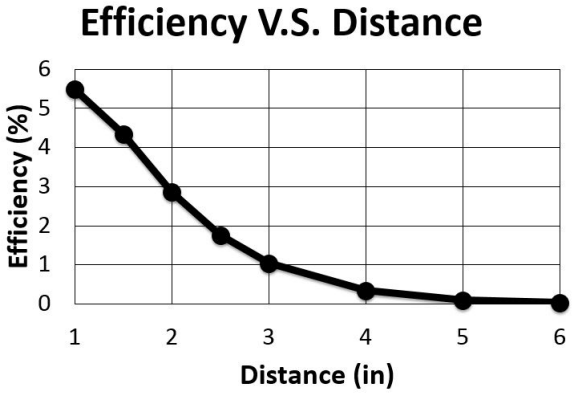

Figure 19: Efficiency versus distance with 100Ω load

After all the components were added to the PCB with charging, additional testing was done. Due to the passive components of the charging chip, the matching resonant frequency of the receiver was altered. Addition capacitance was added to the circuit. Approximately 5nF were needed for an ideal matching resonant frequency of 2.189 MHz. The total capacitance of the receiver circuit now requires 30 nF. Also, the resistive load of the entire circuit gives a much higher voltage to distance ratio.

8.2.2 CHARGING CHIP

Once the battery was connected to the circuit, current draw for the charging circuit was very minimal. This was found by putting the micro-controller into sleep mode, that way only the charging chip would be drawing

current.

Then the charging coils were connected to the circuit. The only way available to check if the battery was charging was to periodically check the voltage. The battery started out at 3.0 volts and after a little over an hour of charging, the battery increased to 3.2 volts meaning the battery was successfully being charged.

8.2.3 BATTERY LIFE

Battery life was determined by adding the average current draw of each active component on the node. The current draw for each component in each of its states is listed below in Figure 20, to evaluate an approximate value for battery life. State times for the CC1101, by far the largest impact on overall battery life, are estimated from the number of packets to be sent and the programmed baud rate. State times for other components are estimated from the speed of serial communication programmed into the micro-controller. This will determine the amount of mAh used per day by the node.

MSP430F2122					
States	**Current draw (uA)**	**State time (s) /day**	**Average current (uA)**		
Active	220	55.66666667	0.141743827		
LPM0	0.7	0	0		
LPM4	0.11	86344.33333	0.109929128		
CC1101					
States	**Current draw (uA)**	**State time (s) /day**		Baud rate	TX Bits/day
TX	16000	16.66666667	3.086419753	9600	160000
RX	16000	16.5	3.055555556		
IDLE	8400	8	0.777777778		
SLEEP w/ WOR	0.5	86358.83333	0.499761767	**Average current (uA)**	
				9.982011883	
MCP7940M				**Expected Battery Life(yrs)**	
States	**Current draw (uA)**	**State time (s) /day**		7.204740794	
Timekeeping	1.2	86396	1.199944444		
I2C Read/Write	350	4	0.016203704		
SST25VF040B					
States	**Current draw (uA)**	**State time (s) /day**			
Standby	20	4	0.000925926		
Active	15000	6	1.041666667		
OFF	0	86390	0		
SHT71					
States	**Current draw (uA)**	**State time (s) /day**			
Sleep	1.1	0	0		
measuring	1000	4.5	0.052083333		
OFF	0	86395.5	0		

Figure 20: Battery life expectancy and current draw

8.3 ENCLOSURE

The enclosure was put through both a weight test to verify that it would be able to stand up to the pressures that will be experienced during the paving of the concrete. For the enclosure to pass the weight test, it needed to be able to withstand about 50 pounds of weight resting on top of it. For the weight test, the enclosure was placed on a flat surface and weights were added one by one until the desired weight was reached. The enclosure was able to hold up over 50 pounds of weight and over 150 pounds when somebody stood on top of it with no signs of cracking due to stress. Therefore, the enclosure passed the weight survivability test.

The enclosure was also subject to a waterproofing test to verify that it would keep wet concrete and water from reaching the electronics within during the concrete pouring and curing processes. After applying a mesh, water was dripped to cover the entire mesh. The mesh was able to keep the water out and there were no signs of leaks. After putting the entire enclosure together, water was poured along the seals that

were made with the epoxy and again, no water was able to penetrate the enclosure. The enclosure was also subject to wet concrete during the communication testing. After retrieving the enclosure from the concrete and opening it to take out the PCB and battery, there were no signs of water or concrete inside of the enclosure. Looking at the enclosure after both of these tests, it is safe to say that the enclosure passed the waterproofing test.

9 CONCLUSION

Future applications can extend to embedded sensors in other concrete structures besides roadways. Wireless networks can be implemented in bridges and underground passages. Monitoring the health of these structures can prevent fatal accidents and provide accurate scheduling for maintenance. Additionally, the sensors can be used to test and refine different concrete composites for better structures. Implementing these node networks becomes more feasible as the project becomes more large scale; the bulk pricing of parts reduces the cost per node, installation and maintenance is cheap, and the structure is less likely to be replaced early which all saves money. Overall, the wireless network offers an economic form of structural supervision.

Appendices

A OPERATIONAL MANUALS

A.1 BASE STATION

Step 1
Before starting the base station, setup all nodes that will be in the network.

Step 2
Establish a connection to the base station, this can be done in numerous ways. An HDMI display and keyboard can be directly plugged into the station. If it has internet access an SSH connection can be established with it.

Step 3
In the base station's terminal, navigate to the project directory for the base station. Run the executable at "src\base_station". The Base Station will automatically perform configuration of the network.

A.2 CHARGING

Step 1
Plug in the DC power source (12V to 20V). If the distance between the transmitter coil and the receiver coil is within 2.4 inches, the power supply voltage should be set to 12V. If the distance is larger (up to 3.2 inches), the power supply voltage should be at 18V.

Step 2
Place the transmitter directly on top of the receiver node.

Step 3
Turn the transmitter on by flipping the switch. An LED on the switch should light up indicating that the transmitter is on.

A.3 ENCLOSURE

Step 1
Lay 2 to 3 layers of nylon mesh over the outside of the sensor opening on the inner box of the enclosure. Take a slightly larger piece of the ripstop nylon patches and put it over the nylon mesh and press down firmly, removing any air bubbles from beneath the patch. Run seam seal along the outside edge of the nylon patch to seal to the enclosure. See Figure 21. Caution: The seam seal is toxic. Avoid directly breathing the fumes that come from the bottle of seam seal.

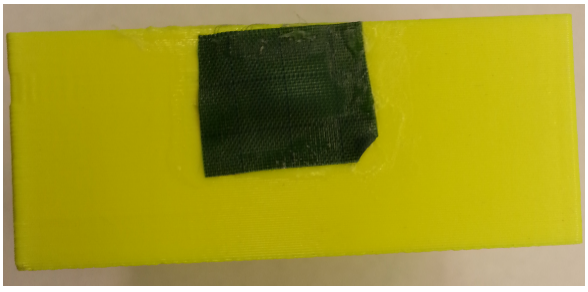

Figure 21: Nylon mesh covering sensor opening

Step 2

Screw #6-32 by 3/8 inch machine screws through each of the four (or three if working with a charging node) mounting holes on the PCB.

Step 3

Lay the battery inside of the inner enclosure between the pegs with the wires of the battery going away from the sensor opening.

Step 4

Place the PCB into the enclosure making sure that the mount for the sensor is facing towards the sensor opening. Push each one of the screws into their respective peg until the PCB and battery are held in place.

Step 5

Place the sensor against the inside of the sensor opening and apply epoxy around the edge of the sensor. This will hold the sensor in place as well as create an airtight seal between the opening and the sensor itself. Wait 5 minutes for the epoxy to set before progressing to step 6. If working with the non-charging node, proceed to step 7. Caution: Avoid breathing the epoxy fumes as much as possible.

Step 6 (For Charging Node Only)

Inside of the inner piece of the enclosure, place the inductive coil above the sensor, along the rim of the inner enclosure. Apply epoxy at the corners to help hold the coil in place. Wait 5 minutes for the epoxy to set before progressing to step 7.

Step 7

Take the outer half of the enclosure and rotate it 90 degrees so that the sensor openings are 90 degrees apart from one another. See Figure 22.

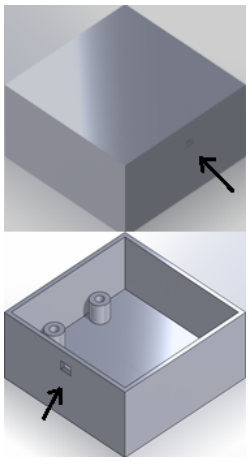

Figure 22: Locations of sensor openings with 90 degree rotation

Step 8

Apply epoxy along the top edge of the walls of the inside piece of the enclosure. Put the inside half of the enclosure into the outer half. **IMPORTANT** - Position the two halves of the enclosure so that the walls that do not contain a sensor opening are as close to one another as possible. This will create an air gap between the walls of the two halves of the enclosure where the sensor openings are located. Hold the two pieces of the enclosure together firmly for about 30 seconds to allow the epoxy to ad-

here to both parts of the enclosure. Let the epoxy sit for 5 minutes before starting step 9. See Figure 23.

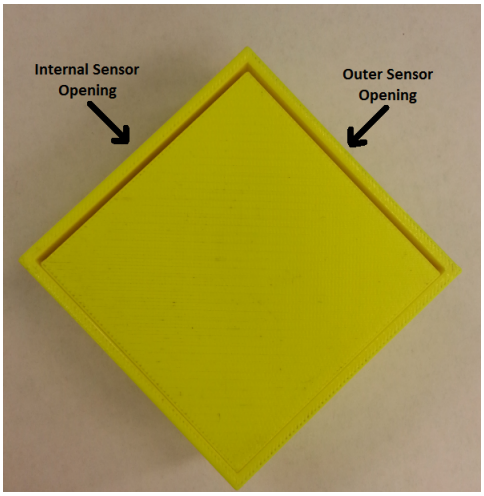

Figure 23: Air gap between enclosure walls

Step 9

Apply epoxy along the bottom of the enclosure where the walls of the two pieces meet as well as along the air gap to create an airtight seal. Scrape away any excess epoxy to create a smooth bottom. Let the enclosure sit for 5 minutes while the epoxy hardens before continuing with any work on the node itself.

B Alternative Project Plans/Designs

There was another design that was considered for the communications and power charging systems. One idea was that a system of receiver nodes would be placed in the ground beneath the concrete before pouring. See Figure 24 for alternate design concept. These nodes would be able to be wired to a constant power supply as well as the base station. With this design, there could be constant power transferred to the sensing nodes allowing for infinite charging times. The receiver nodes would be paired with a sensor node and would receive the data being sent. This would allow for smaller distances needing to be penetrated by both the charging and communication systems. The buried nodes having communication wires would greatly increase the success rate of data transmission as well as allow the transceivers to use lower power for sending data. The ground in which the receiving nodes would be placed is extremely hard and would require digging equipment to implement this system. This is the main reason that the design was not implemented because it would be time consuming, expensive, and difficult.

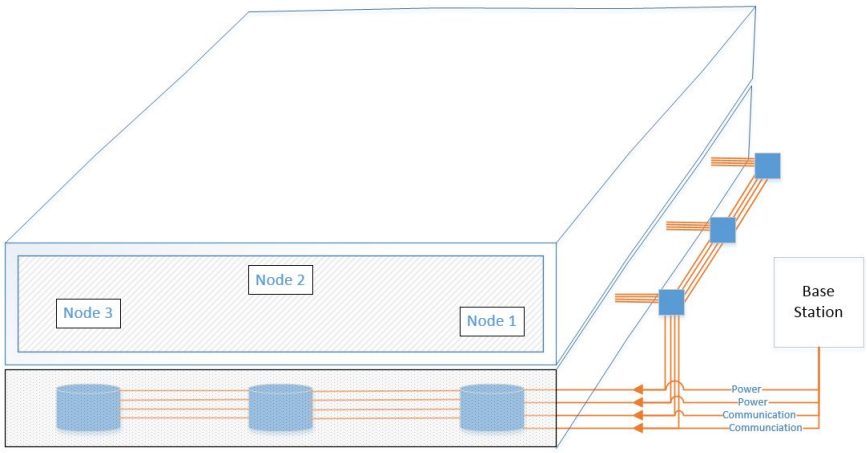

Figure 24: Concept sketch of an alternate way to charge and retrieve data

Another idea for energy harvesting was to implement a radio frequency patch antenna that would receive radio waves from a transmission antenna to generate energy for the nodes. This as well as the inductive coils were researched and tested before a design was chosen for the charging system. There were a few issues with the RF antenna that caused it to be taken out of the design. Firstly, the patch antenna that would be needed for the node was too large. Second, the inductive coils could be reduced in size which would cause less flux to penetrate the receiving coil, however, reducing the antenna size would change the harmonic frequency of the antenna. Both of these issues helped in the decision to implement inductive coils as the energy harvesting system for the nodes.

C Other Considerations or Mistakes

C.1 Charging Chip

During initial testing of the charging circuit, the prototype charging circuit was used to charge the testing battery. Ten minutes later, there was a faint smell of burning electronics and then the chip sparked and caught fire momentarily. The circuit was immediately disconnected and completely re-designed. To prevent further fire hazards, Linear Technologies was consulted to find the best LI-ION charging IC to fit our needs.

During Testing of the charging circuit, it was discovered that the charging IC was placing a float voltage of 4.2 volts over the battery. The circuit was designed to have a float voltage of 3.7V. The conclusion was that an incorrect charging chip IC was received. There are two versions of the charging IC, with one version having a predetermined float voltage of 4.2 volts.

C.2 RF Charging

During the design phase, battery charging via RF waves in the 915MHz band was considered[2].The receiving antenna was a quasi-microstrip antenna. The antenna was composed of two conducting, rectangular planes -with small center holes- separated by an air gap.The top plane was designated the conducting plane and the bottom the ground. A stripped coaxial cable was inserted within the holes while the outer ground wires were soldered to the bottom. The conducting portion of the cable was soldered to the top plane. The end of the cable, with sma connector intact, would lead to the battery charging circuit.
Inductive charging was decided over RF charging on the merit of efficiency. Waves at higher frequencies attenuate faster traveling through concrete; in order to charge the battery, the charging circuit requires a threshold current in order to operate.

C.3 Soldering

Size constraints for the node, required the use of surface mount components, which in turn required re-flow soldering for assembly. None of the members of the group had any experience with this process or with components this small, so mistakes in assembly occurred a number of times and set back development. The majority of theses mistakes were from applying too much solder paste. This caused the components to shift in the oven and sometimes even going vertical, pins becoming bridged, and aligning the QFN integrated circuits.

D Team Contributions

The following Figure 25 is a representation of each individuals total contributed hours throughout the year of senior design. Figure 26 shows the total hours per individual along with the total amount of time.

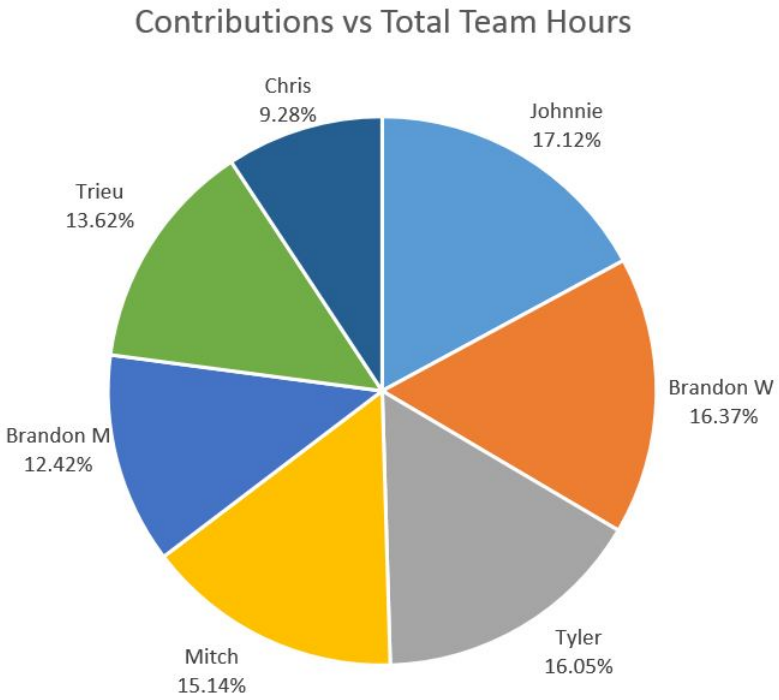

Figure 25: Individual Contributed Hours

Johnnie	Brandon	Tyler	Mitch	Brandon	Trieu	Chris	Total
160.5	153.5	150.5	142	116.5	127.75	87	937.75
17.12%	16.37%	16.05%	15.14%	12.42%	13.62%	9.28%	

Figure 26: Individual Contributed Hours and Total Time

E CODE

E.1 MAIN PCB CODING

```
/*
 *       main.c
 */

#include <msp430.h>
#include "spi.h"
#include "cc1101.h"
#include "flash.h"
#include "microcontroller.h"
#include "sht10.h"

#define TX 1

#define tx_size 62

volatile unsigned char timer_flag = 0;
volatile unsigned char rx_flag = 0;
volatile unsigned char temp, humidity;
volatile int beacon_count;


int main(void) {

        WDTCTL = WDTPW + WDTHOLD;

        mcu_setup();
        spi_setup();
        //flash_setup();
        cc1101_config(0x02, 0x00);
        CC1101_set_output_power(CC_PWR_7);

        temp = 0;
        humidity = 0;
        beacon_invterval = 5000;

        led_flash();
        led_flash();
        led_flash();

        //Send packets

        timer_flag = 0;
        beacon_count = 0;

        buffer[0] = tx_size;

        //Broadcast packet
        buffer[1] = 0xFF;
```

```
int i;
for (i = 2; i < tx_size + 1; i++) {
        buffer[i] = i - 1;
}

CC1101_strobe(CC_SFTX);

while (1) {

        //If RX overflow
        if ((CC1101_strobe(CC_SNOP) & 0x70) == 0x60) {
                CC1101_strobe(CC_SIDLE);
                CC1101_strobe(CC_SFRX);
        }

        CC1101_strobe(CC_SRX);

        if (timer_flag) {

                //Get temperature and humidity measurements
                SHT_ON;
                temp = read_temperature();
                humidity = read_humidity();
                SHT_OFF;

                CC1101_wake();

                //Packet ID as current timer value

                unsigned char id = TA0R & 0x0F;
                buffer[2] = id;

                buffer[3] = temp;
                buffer[4] = humidity;

                cc1101_send_packet(buffer, tx_size);
                led_flash();

                beacon_count++;
                timer_flag = 0;

                rx_packet_ids[rx_index] = id;
                rx_index++;
                if(rx_index > 15){
                        rx_index = 0;
                }


        }else if(rx_flag){
```

```
                        //Receive packet from CC1101
                        cc1101_rcv_packet(buffer, &rx_size);
                        rx_flag = 0;

                        unsigned char id_in = buffer[1];
                        rx_packet_ids[rx_index] = id_in;
                        rx_index++;
                        if(rx_index > 15){
                                rx_index = 0;
                        }

                        int i;
                        char received = 0;

                        for(i = 0; i < 16; i++){
                                if(id_in == rx_packet_ids[i]){
                                        received = 1;
                                }
                        }

                        if(received == 0){
                                //Check if this packet has been to this node, if not, forwa
                                int j;
                                for(j = rx_size-1; j > 0; j--){
                                        buffer[j] = buffer[j-1];
                                }
                                buffer[0] = tx_size;
                                cc1101_send_packet(buffer, tx_size);
                        }
                }

                _bis_SR_register(LPM1_bits + GIE);      //Put uC to sleep
        }

        while (1);
}

// Port 1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void Port_1_ISR(void)
{

        P1IFG &= ~GDO2;
        rx_flag = 1;

}

#pragma vector = TIMER0_A0_VECTOR
__interrupt void TIMERA0_ISR() {
```

```
        static int timer_count = 0;
        timer_count++;

        if (timer_count > beacon_invterval) {
                timer_count = 0;
                timer_flag = 1;
                _bic_SR_register_on_exit(LPM1_bits);
        } else {
                timer_flag = 0;
        }

}
```

```
/*
 *       microcontroller.h
 */


#ifndef _MICROCONTROLLER_H_
#define _MICROCONTROLLER_H_

#define LED_PIN BIT3
#define LED_ON P1OUT |= LED_PIN
#define LED_OFF P1OUT &= ~LED_PIN


#define WDT_HOLD WDTCTL = WDTPW + WDTHOLD
#define WDT_RESET WDTCTL = WDTPW + WDTCNTCL

#define FLASH_ON P2OUT |= BIT3
#define FLASH_OFF P2OUT &= BIT3

unsigned char rx_packet_ids[16];
int rx_index = 0;

volatile unsigned int beacon_invterval;

void mcu_setup();

/* Sleep mode with interrupt wakeup */
void mcu_sleep_gie();

/* Wait for interrupt wakeup */
void mcu_wait_gie();

void led_flash();


#endif /* MICROCONTROLLER_H_ */
```

```
/*
 *    Microcontroller.c
 */

#include <msp430.h>
#include "microcontroller.h"
#include "sht10.h"
#include "cc1101.h"

#define INTERRUPT_PRAGMA

void mcu_setup() {
        /* Stop watchdog timer */
        WDTCTL = WDTPW | WDTHOLD;

        /* 1MHZ clock */
        BCSCTL1 = CALBC1_1MHZ;
        DCOCTL = CALDCO_1MHZ;

        /*      TimerA config
         *      -Source from ACLK
         *      -Up mode
         *      -Enable TimerA interrupt
         */
        TACCR0 = 32768;
        TACTL = TASSEL_2 + MC_1;
        TACCTL0 |= CCIE;

        P1DIR |= LED_PIN;
        P1OUT = 0;

        P2DIR |= SCL_PIN + SHT_VCC;
        P2REN |= SDA_PIN;

        P2DIR |= BIT3;
        P2OUT |= BIT3;

        P1IE |= GDO2;
        P1IFG &= ~GDO2;

        int i;
        for(i = 0; i < 16; i++){
                rx_packet_ids = 0;
        }

        /* Enable interrupt */
        _bis_SR_register(GIE);
}

void led_flash() {
```

```
        volatile char debug = 1;

        if(!debug){
                return;
        }

        int j = 6;
        for (; j != 0; j--) {
                volatile unsigned int i;
                P1OUT ^= LED_PIN;
                _delay_cycles(30000);
        }
        P1OUT &= ~LED_PIN;
}
```

```
/*
 * cc1101.h
 *
 *  Created on: Dec 3, 2014
 *      Author: mdbalke
 */

#ifndef CC_1101_H_
#define CC_1101_H_


#define CS_ENABLE (P1OUT &= ~CS)
#define CS_DISABLE (P1OUT |= CS)

#define CC1101_MAX_FIFO 64

//Definition for SPI lines on port 1
//#define CS BIT5
#define GDO0 BIT6
#define GDO2 BIT7

#define MAX_RXFIFO 64
#define MAX_TXFIFO 64

// CC1101 SPI Header: (R/!W) + (Burst) + (A5–A0)
#define CC_HEADER_RW    BIT7 // Read/!Write
#define CC_HEADER_BURST BIT6 // Burst/!Single

#define CC_STATUS_FIFO(x)       (x & 0b00001111)
#define CC_STATUS_STATE(x)      (x & 0b01110000)

/*
 * Single TX FIFO       0x3F    0b00 111111
 * Burst TX FIFO        0x7F    0b01 111111
 * Single RX FIFO       0xBF    0b10 111111
 * Burst RX FIFO        0xFF    0b11 111111
 */

// PATABLE (0 dBm output power)
//char paTable[] = {0x51};
//char paTableLen = 1;

volatile unsigned char buffer[64];
volatile unsigned char status_1, status_2;
volatile unsigned char status_reg;



// Configuration Registers
#define CC_IOCFG2       0x00        // GDO2 output pin configuration
#define CC_IOCFG1       0x01        // GDO1 output pin configuration
```

```
#define CC_IOCFG0       0x02        // GDO0 output pin configuration
#define CC_FIFOTHR      0x03        // RX FIFO and TX FIFO thresholds
#define CC_SYNC1        0x04        // Sync word, high byte
#define CC_SYNC0        0x05        // Sync word, low byte
#define CC_PKTLEN       0x06        // Packet length
#define CC_PKTCTRL1     0x07        // Packet automation control
#define CC_PKTCTRL0     0x08        // Packet automation control
#define CC_ADDR         0x09        // Device address
#define CC_CHANNR       0x0A        // Channel number
#define CC_FSCTRL1      0x0B        // Frequency synthesizer control
#define CC_FSCTRL0      0x0C        // Frequency synthesizer control
#define CC_FREQ2        0x0D        // Frequency control word, high byte
#define CC_FREQ1        0x0E        // Frequency control word, middle byte
#define CC_FREQ0        0x0F        // Frequency control word, low byte
#define CC_MDMCFG4      0x10        // Modem configuration
#define CC_MDMCFG3      0x11        // Modem configuration
#define CC_MDMCFG2      0x12        // Modem configuration
#define CC_MDMCFG1      0x13        // Modem configuration
#define CC_MDMCFG0      0x14        // Modem configuration
#define CC_DEVIATN      0x15        // Modem deviation setting
#define CC_MCSM2        0x16        // Main Radio Cntrl State Machine config
#define CC_MCSM1        0x17        // Main Radio Cntrl State Machine config
#define CC_MCSM0        0x18        // Main Radio Cntrl State Machine config
#define CC_FOCCFG       0x19        // Frequency Offset Compensation config
#define CC_BSCFG        0x1A        // Bit Synchronization configuration
#define CC_AGCCTRL2     0x1B        // AGC control
#define CC_AGCCTRL1     0x1C        // AGC control
#define CC_AGCCTRL0     0x1D        // AGC control
#define CC_WOREVT1      0x1E        // High byte Event 0 timeout
#define CC_WOREVT0      0x1F        // Low byte Event 0 timeout
#define CC_WORCTRL      0x20        // Wake On Radio control
#define CC_FREND1       0x21        // Front end RX configuration
#define CC_FREND0       0x22        // Front end TX configuration
#define CC_FSCAL3       0x23        // Frequency synthesizer calibration
#define CC_FSCAL2       0x24        // Frequency synthesizer calibration
#define CC_FSCAL1       0x25        // Frequency synthesizer calibration
#define CC_FSCAL0       0x26        // Frequency synthesizer calibration
#define CC_RCCTRL1      0x27        // RC oscillator configuration
#define CC_RCCTRL0      0x28        // RC oscillator configuration
#define CC_FSTEST       0x29        // Frequency synthesizer cal control
#define CC_PTEST        0x2A        // Production test
#define CC_AGCTEST      0x2B        // AGC test
#define CC_TEST2        0x2C        // Various test settings
#define CC_TEST1        0x2D        // Various test settings
#define CC_TEST0        0x2E        // Various test settings

// Status registers – read only
#define CC_PARTNUM      0x30        // Part number
#define CC_VERSION      0x31        // Current version number
#define CC_FREQEST      0x32        // Frequency offset estimate
#define CC_LQI          0x33        // Demodulator estimate for link quality
```

```
#define CC_RSSI             0x34        // Received signal strength indication
#define CC_MARCSTATE        0x35        // Control state machine state
#define CC_WORTIME1         0x36        // High byte of WOR timer
#define CC_WORTIME0         0x37        // Low byte of WOR timer
#define CC_PKTSTATUS        0x38        // Current GDOx status and packet status
#define CC_VCO_VC_DAC       0x39        // Current setting from PLL cal module
#define CC_TXBYTES          0x3A        // Underflow and # of bytes in TXFIFO
#define CC_RXBYTES          0x3B        // Overflow and # of bytes in RXFIFO
#define CC_NUM_RXBYTES      0x7F        // Mask "# of bytes" field in _RXBYTES

// Strobe commands
#define CC_SRES             0x30        // Reset chip.
#define CC_SFSTXON          0x31        // Enable/calibrate freq synthesizer
#define CC_SXOFF            0x32        // Turn off crystal oscillator.
#define CC_SCAL             0x33        // Calibrate freq synthesizer & disable
#define CC_SRX              0x34        // Enable RX.
#define CC_STX              0x35        // Enable TX.
#define CC_SIDLE            0x36        // Exit RX / TX
#define CC_SAFC             0x37        // AFC adjustment of freq synthesizer
#define CC_SWOR             0x38        // Start automatic RX polling sequence
#define CC_SPWD             0x39        // Enter pwr down mode when CSn goes high
#define CC_SFRX             0x3A        // Flush the RX FIFO buffer.
#define CC_SFTX             0x3B        // Flush the TX FIFO buffer.
#define CC_SWORRST          0x3C        // Reset real time clock.
#define CC_SNOP             0x3D        // No operation.

// Output power definitions
#define CC_PWR_NEG_30       0x01
#define CC_PWR_NEG_20       0x02
#define CC_PWR_NEG_15       0x03
#define CC_PWR_NEG_10       0x04
#define CC_PWR_0            0x05
#define CC_PWR_5            0x06
#define CC_PWR_7            0x07
#define CC_PWR_10           0x08

#define CC_PATABLE          0x3E

unsigned char CC1101_reg_write(unsigned char address, unsigned char data);
unsigned char CC1101_burst_reg_write(unsigned char starting_address, unsigned char *data,
unsigned char CC1101_reg_read(unsigned char address);
unsigned char CC1101_strobe(unsigned char strobe);
unsigned char CC1101_read_status_register(unsigned char address);

unsigned char cc1101_send(int num_bytes);
unsigned char cc1101_wait_for_packet();

void cc1101_config(unsigned char device_address, unsigned char channel_number);
void cc1101_send_packet(unsigned char *data, int num_bytes);
unsigned char cc1101_rcv_packet(unsigned char *data, int *num_bytes);
```

```
unsigned char CC1101_sleep_wake_on_radio();

unsigned char CC1101_burst_reg_read(unsigned char starting_address, unsigned char *data, in

void CC1101_set_output_power(unsigned char pwr);

void CC1101_sleep();
void CC1101_wake();

unsigned char checksum(unsigned char *ptr, int length);


#endif /* CC_1101_H_ */
```

```
/*
 *      cc1101.c
 */


#include <msp430.h>
#include "spi.h"
#include "cc1101.h"
#include "microcontroller.h"

#define HEADER_READ(x) (x | CC_HEADER_RW)
#define HEADER_WRITE(x) (x & ~CC_HEADER_RW)
#define HEADER_BURST(x) (x | CC_HEADER_BURST)
#define HEADER_SINGLE(x) (x & ~CC_HEADER_BURST)

unsigned char CC1101_reg_write(unsigned char address, unsigned char data) {
        unsigned char header = (address & ~CC_HEADER_RW) & ~CC_HEADER_BURST;

        volatile unsigned char status;

        CC1101_SELECT;

        //TODO Wait for CC1101 to be ready
        spi_tx(header);
        status = spi_tx(data);

        CC1101_DESELECT;

        return status;
}

unsigned char CC1101_burst_reg_write(unsigned char starting_address,
                unsigned char *data, int num_bytes) {
        unsigned char header = (starting_address & ~CC_HEADER_RW) | CC_HEADER_BURST;

        volatile unsigned char status;
        int i;

        CC1101_SELECT;
        spi_tx(header);
        for (i = 0; i < num_bytes; i++){
                status = spi_tx(data[i]);
        }
        CC1101_DESELECT;

        return status;
}

unsigned char CC1101_reg_read(unsigned char address) {
        unsigned char header = (address | CC_HEADER_RW) & ~CC_HEADER_BURST;
```

```
        volatile unsigned char status;

        CC1101_SELECT;

        spi_tx(header);
        status = spi_tx(0x00);

        CC1101_DESELECT;

        return status;
}

unsigned char CC1101_burst_reg_read(unsigned char starting_address,
                unsigned char *data, int num_bytes) {
        unsigned char header = (starting_address | CC_HEADER_RW) | CC_HEADER_BURST;

        volatile unsigned char status;
        int i;

        CC1101_SELECT;
        status = spi_tx(header);
        for (i = 0; i < num_bytes; i++){
                data[i] = spi_tx(0x00);
        }
        CC1101_DESELECT;

        return status;
}

unsigned char CC1101_strobe(unsigned char strobe) {
        volatile unsigned char status;

        CC1101_SELECT;

        status = spi_tx(strobe);

        CC1101_DESELECT;

        return status;
}

void cc1101_config(unsigned char device_address, unsigned char channel_number) {
        /* Set Slave Select as output and set high */
        P1DIR &= ~GDO0; // GDO0 input
        P1DIR &= ~GDO2; // GDO2 input

        _delay_cycles(1000);

        CC1101_strobe(CC_SRES);

        _delay_cycles(1000);
```

```
        // Write register settings
        CC1101_reg_write(CC_IOCFG2, 0x01); // GDO2 output pin config.                //
        CC1101_reg_write(CC_IOCFG0, 0x06); // GDO0 output pin config.                //
        CC1101_reg_write(CC_PKTLEN, 64); // Packet length.
        CC1101_reg_write(CC_PKTCTRL1, 0x03); // Packet automation control.  //0x05
        CC1101_reg_write(CC_PKTCTRL0, 0x01); // Packet automation control.          //
        CC1101_reg_write(CC_ADDR, device_address); // Device address.
        CC1101_reg_write(CC_CHANNR, channel_number); // Channel number.
        CC1101_reg_write(CC_FSCTRL1, 0x0B); // Freq synthesizer control.
        CC1101_reg_write(CC_FSCTRL0, 0x00); // Freq synthesizer control.
        CC1101_reg_write(CC_FREQ2, 0x10); // Freq control word, high byte
        CC1101_reg_write(CC_FREQ1, 0xA7); // Freq control word, mid byte.
        CC1101_reg_write(CC_FREQ0, 0x62); // Freq control word, low byte.
        CC1101_reg_write(CC_MDMCFG4, 0x2D); // Modem configuration.
        CC1101_reg_write(CC_MDMCFG3, 0x3B); // Modem configuration.
        CC1101_reg_write(CC_MDMCFG2, 0x73); // Modem configuration.
        CC1101_reg_write(CC_MDMCFG1, 0x22); // Modem configuration.
        CC1101_reg_write(CC_MDMCFG0, 0xF8); // Modem configuration.
        CC1101_reg_write(CC_DEVIATN, 0x00); // Modem dev (when FSK mod en)
        CC1101_reg_write(CC_MCSM1, 0x3F); // MainRadio Cntrl State Machine
        CC1101_reg_write(CC_MCSM0, 0x18); // MainRadio Cntrl State Machine
        CC1101_reg_write(CC_FOCCFG, 0x1D); // Freq Offset Compens. Config
        CC1101_reg_write(CC_BSCFG, 0x1C); //  Bit synchronization config.
        CC1101_reg_write(CC_AGCCTRL2, 0xC7); // AGC control.
        CC1101_reg_write(CC_AGCCTRL1, 0x00); // AGC control.
        CC1101_reg_write(CC_AGCCTRL0, 0xB2); // AGC control.
        CC1101_reg_write(CC_FREND1, 0xB6); // Front end RX configuration.
        CC1101_reg_write(CC_FREND0, 0x10); // Front end RX configuration.
        CC1101_reg_write(CC_FSCAL3, 0xEA); // Frequency synthesizer cal.
        CC1101_reg_write(CC_FSCAL2, 0x0A); // Frequency synthesizer cal.
        CC1101_reg_write(CC_FSCAL1, 0x00); // Frequency synthesizer cal.
        CC1101_reg_write(CC_FSCAL0, 0x11); // Frequency synthesizer cal.
        CC1101_reg_write(CC_FSTEST, 0x59); // Frequency synthesizer cal.
        CC1101_reg_write(CC_TEST2, 0x88); // Various test settings.
        CC1101_reg_write(CC_TEST1, 0x31); // Various test settings.
        CC1101_reg_write(CC_TEST0, 0x0B); // Various test settings.
        CC1101_reg_write(CC_FIFOTHR, 14); // RX/TX FIFO capacity trigger          //

        //TODO Device checking
        //CC1101_strobe(CC_SFRX);

}

/*
 * Sends a packet from data buffer
 */
void cc1101_send_packet(unsigned char *data, int num_bytes) {

        //TODO Handle num_bytes > MAX_TXFIFO - 2 (for length and device addr)
```

```
        //TODO Handle packet size and device address as first two bytes

        CC1101_burst_reg_write(0x3F, data, num_bytes);
        status_2 = CC1101_read_status_register(CC_TXBYTES);
        CC1101_strobe(CC_STX);

        while (!(P1IN & GDO0));
        while (P1IN & GDO0);
        if((CC1101_strobe(CC_SNOP) & 0x07) == 0x07){
                CC1101_strobe(CC_SFTX);
        }
}

/*
 * Reads buffer from RX_FIFO after checking that there is data to be read
 *
 * return: 0xFF if RX_FIFO overflow
 *                 0 if data was read from buffer and CRC-pass
 *                 1 if data was read from buffer but CRC-fail
 */
unsigned char cc1101_rcv_packet(unsigned char *data, int *num_bytes) {


        unsigned char pktlen = CC1101_reg_read(0xBF);
        //unsigned char pktlen = CC1101_read_status_register(CC_RXBYTES) & 0x7F;

        *num_bytes = pktlen;
        CC1101_burst_reg_read(0xFF, data, pktlen);

        return 0;

}

/*
 * Reads the value of a read-only status register
 *
 */
unsigned char CC1101_read_status_register(unsigned char address) {

        unsigned char header = address | CC_HEADER_RW | CC_HEADER_BURST;

        volatile unsigned char data;

        CC1101_SELECT;

        spi_tx(header);
        data = spi_tx(0x00);

        CC1101_DESELECT;

        return data;
```

```
}

/*
 * Puts CC1101 to sleep
 *
 */
void CC1101_sleep() {

        //Set GDO pins as high impedance to save power
        //CC1101_reg_write(CC_IOCFG2, 0x2E);
        //CC1101_reg_write(CC_IOCFG0, 0x2E);

        CC1101_strobe(CC_SPWD); //Put CC1101 to sleep

}

/*
 * Wakes CC1101 from sleep mode and restores settings
 *
 */
void CC1101_wake() {

        CC1101_SELECT;
        while (P1IN & BIT7);
        CC1101_DESELECT;


        //Restore GDO states
        //CC1101_reg_write(CC_IOCFG2, 0x01);
        //CC1101_reg_write(CC_IOCFG0, 0x06);


        CC1101_strobe(CC_SIDLE);
        CC1101_strobe(CC_SFRX);

}

/*
 * Sets CC1101 output power by writing to first byte of PATABLE
 *      -Other bytes of PATABLE are ignored as ramp up/down are not used
 *
 */
void CC1101_set_output_power(unsigned char pwr){

        switch(pwr){
        case CC_PWR_NEG_30:
                CC1101_reg_write(CC_PATABLE, 0x12);
                break;
        case CC_PWR_NEG_20:
                CC1101_reg_write(CC_PATABLE, 0x0E);
```

```
                break;
        case CC_PWR_NEG_15:
                CC1101_reg_write(CC_PATABLE, 0x1D);
                break;
        case CC_PWR_NEG_10:
                CC1101_reg_write(CC_PATABLE, 0x34);
                break;
        case CC_PWR_0:
                CC1101_reg_write(CC_PATABLE, 0x60);
                break;
        case CC_PWR_5:
                CC1101_reg_write(CC_PATABLE, 0x84);
                break;
        case CC_PWR_7:
                CC1101_reg_write(CC_PATABLE, 0xC8);
                break;
        case CC_PWR_10:
                CC1101_reg_write(CC_PATABLE, 0xC0);
                break;
        default:
                CC1101_reg_write(CC_PATABLE, 0x60);
                break;
        }
}



/*
 *  Used to generate and check 8-bit checksum
 *     -Pass buffer without checksum to return checksum value
 *         -Pass buffer with checksum to return 0 on checksum pass
 */
unsigned char checksum(unsigned char *ptr, int length){
        unsigned char value = 0;
        while (length-- != 0)
                value -= *ptr++;
        return value;
}
```

```
/*
 *       spi.h
 */

#ifndef SPI_H_
#define SPI_H_

#define CC1101_CS BIT5
#define CC1101_SELECT P1OUT &= ~CC1101_CS
#define CC1101_DESELECT P1OUT |= CC1101_CS

#define FLASH_CS BIT2
#define FLASH_SELECT P1OUT &= ~FLASH_CS
#define FLASH_DESELECT P1OUT |= FLASH_CS

#define MISO BIT5
#define MOSI BIT4
#define SCK BIT0

#define SPI_READ (P3IN & MISO)

//void spi_setup(void (*spi_rx)(char));
void spi_setup();

/* SPI Transmit in Low Power Mode, Interrupt Unsafe */
unsigned char spi_tx_lpm_iu(unsigned char tx);

/* SPI Transmit in Active Power mode (interrupt safe) */
unsigned char spi_tx(unsigned char tx);

#endif /* SPI_H_ */
```

```
/*
 *      spi.c
 */

#include <msp430.h>
#include "microcontroller.h"
#include "spi.h"

// Slave mode ifdef, else Master mode
//#define SLAVE_SPI

volatile char interrupt_rx; // Temporary register for storing rx from spi_interrupt

void spi_setup() {

        P1DIR |= CC1101_CS + FLASH_CS + BIT1;


        P3SEL |= BIT0 + BIT5 + BIT4;
        P3DIR |= BIT6;

        FLASH_DESELECT;
        CC1101_DESELECT;

        UCA0CTL0 |= UCCKPH + UCMSB + UCMST + UCSYNC; // 3-pin, 8-bit SPI master
        UCA0CTL1 |= UCSSEL_2;                        // SMCLK
        UCA0BR0 |= 0x02;
        UCA0BR1 = 0;                                 //
        UCA0MCTL = 0;
        UCA0CTL1 &= ~UCSWRST;                       // **Initialize USCI state machine**

        P1OUT |= BIT1;
        P3OUT |= BIT6;
}

unsigned char spi_tx(unsigned char tx) {

        unsigned char out = 0;

        IFG2 &= ~UCA0RXIFG;
        UCA0TXBUF = tx;
        while (!(IFG2 & UCA0RXIFG));
        out = UCA0RXBUF;
        return out;
}

unsigned char spi_tx_lpm_iu(unsigned char tx) {

        unsigned char out = 0;

        IFG2 &= ~UCA0RXIFG;
```

```
        UCA0TXBUF = tx;
        while (!(IFG2 & UCA0RXIFG));
        out = UCA0RXBUF;
        return out;

}
```

```
/*
 * sht10.h
 */

#ifndef SHT71_H_
#define SHT71_H_



#define SDA_PIN BIT1
#define SCL_PIN BIT0
#define SHT_VCC BIT2

#define LED_PIN BIT3
#define LED_ON P1OUT |= LED_PIN
#define LED_OFF P1OUT &= ~LED_PIN

#define SHT_ON P2OUT |= SHT_VCC
#define SHT_OFF P2OUT &= ~SHT_VCC

#define DATA_HIGH P2OUT |= SDA_PIN
#define DATA_LOW P2OUT &= ~SDA_PIN
#define DATA_READ (P2IN & SDA_PIN)

#define DATA_IN SHT_data_in()
#define DATA_OUT P2DIR |= SDA_PIN

#define SCL_HIGH P2OUT |= SCL_PIN
#define SCL_LOW P2OUT &= ~SCL_PIN
#define SCL_OUT P2DIR |= SCL_PIN

float read_humidity();
float read_humidity_raw();
float read_temperature();
int read_temperature_raw();
void send_SHT_command(int command);
int read_two_bytes_SHT();
void skipCrcSHT();
void send_byte_SHT(int command);
unsigned char read_byte_SHT();

void SHT_data_in();

void led_flash();


#endif
```

```
/*
 * sht10.c
 */

#include <msp430.h>
#include "sht10.h"
#include "microcontroller.h"

float read_humidity() {

    float linear_value;
    float raw_humidity;

    /*
     * Conversion constants as defined in the datasheet
     *  for linearlizing sensor value
     *
     *  H(linear) = C1 + C2*H(raw) + C3*H(raw)^2
     */
    const float C1 = -2.0468;
    const float C2 = 0.0367;
    const float C3 = .0000015955;

    raw_humidity = read_humidity_raw();

    linear_value = C1 + C2 * raw_humidity + C3 * raw_humidity * raw_humidity;

    return linear_value;
}

float read_humidity_raw() {

    int out;

    const int command = 0b00000101;

    send_SHT_command(command);

    DATA_IN;

    while (DATA_READ) {
        _delay_cycles(10);
    }

    out = read_two_bytes_SHT();

    skipCrcSHT();

    return out;
}
```

```
float read_temperature() {

    int raw_value;
    float temperature;

    const float D1 = -40.0;
    const float D2 = 0.018;

    raw_value = read_temperature_raw();

    temperature = (raw_value * D2) + D1;

    return temperature;
}

int read_temperature_raw() {

    int out;

    const int command = 0b00000011;

    send_SHT_command(command);

    DATA_IN;

    int timeout = 0;

    while (DATA_READ) {
        _delay_cycles(10);
        timeout += 1;
        if(timeout > 1000000){          //Timeout if sensor has malfunctioned
                return 0;
        }
    }

    out = read_two_bytes_SHT();

    skipCrcSHT();

    return out;
}

void send_SHT_command(int command) {

    int ack;

    DATA_OUT;
    SCL_OUT;

    //Send start sequence
    DATA_HIGH;
```

```
    SCL_HIGH;
    DATA_LOW;
    SCL_LOW;
    SCL_HIGH;
    DATA_HIGH;
    SCL_LOW;

    //Send command byte
    send_byte_SHT(command);

    //Look for ack
    SCL_HIGH;
    DATA_IN;
    ack = DATA_READ;

    if (ack != 0) {
        //no ack
    }

    SCL_LOW;
    ack = DATA_READ;
    if (ack == 0) {
        //ack error
    }
}

void send_byte_SHT(int command) {
    int i;
    for (i = 7; i >= 0; i--) {
        if (command & (1 << i)) {
            DATA_HIGH;
        } else {
            DATA_LOW;
        }
        SCL_HIGH;
        SCL_LOW;
    }

}

int read_two_bytes_SHT() {

    int out;

    DATA_IN;

    out = read_byte_SHT();
    out = out << 8;

    //Send ack to sensor for first byte
    DATA_OUT;
```

```
    DATA_HIGH;
    DATA_LOW;
    SCL_HIGH;
    SCL_LOW;

    DATA_IN;
    out |= read_byte_SHT();

    return out;

}

unsigned char read_byte_SHT() {

    unsigned char out = 0;
    int i;
    for (i = 7; i >= 0; i--) {
        SCL_HIGH;
        if (DATA_READ) {
            out |= (1 << i);
        }
        SCL_LOW;
    }
    return out;
}

void skipCrcSHT() {

    DATA_OUT;
    SCL_OUT;

    DATA_HIGH;
    SCL_HIGH;
    SCL_LOW;

}

void SHT_data_in() {
    P2DIR &= ~SDA_PIN;
    P2REN |= SDA_PIN;
}
```

```
/*
 * flash.h
 */

#ifndef FLASH_H_
#define FLASH_H_

unsigned char flash_status_reg();
void flash_WREN();
void flash_WRSR(unsigned char value);

unsigned char flash_read(unsigned long addr);
void flash_read_buffer(unsigned long starting_addr, unsigned int num_bytes, unsigned char *
void flash_write(unsigned long addr, unsigned char value);
void wait_flash_busy();
void flash_reset();

unsigned char flash_setup();

#endif /* FLASH_H_ */
```

```
/*
 * flash.c
 */

#include <msp430.h>
#include "microcontroller.h"
#include "spi.h"
#include "flash.h"

/*
 * Reads the value of the SST25VF040B status register
 *
 */
unsigned char flash_status_reg() {

        unsigned char out = 0;
        FLASH_SELECT;
        _delay_cycles(100);

        spi_tx(0x05);
        out = spi_tx(0);
        FLASH_DESELECT;
        return out;
}

/*
 * Enables the write-enable latch on the SST25VF040B
 *
 */
void flash_WREN() {

        FLASH_SELECT;
        spi_tx(0x06);
        FLASH_DESELECT;
}

/*
 * Writes to the SST25VF040B status register
 *
 */
void flash_WRSR(unsigned char value) {

        flash_WREN();
        FLASH_SELECT;
        spi_tx(0x01);
        spi_tx(value);
        FLASH_DESELECT;
}

/*
 * Reads one byte stored in memory
```

```
 *
 */
unsigned char flash_read(unsigned long addr) {

        unsigned char out = 0;

        FLASH_SELECT;
        spi_tx(0x03);
        spi_tx((addr & 0xFFFFFF) >> 16);
        spi_tx((addr & 0xFFFF) >> 8);
        spi_tx(addr & 0xFF);
        out = spi_tx(0);
        FLASH_DESELECT;
        return out;
}

/*
 * Reads multiple bytes from flash memory
 *    - a maximum of 128 bytes can be read at a time
 * TODO: consider writing bytes to global buffer instead of passing pointer
 *
 */
void flash_read_buffer(unsigned long starting_addr, unsigned int num_bytes,
                unsigned char *buffer) {

        unsigned int i;

        FLASH_SELECT;
        spi_tx(0x03);
        spi_tx((starting_addr & 0xFFFFFF) >> 16);
        spi_tx((starting_addr & 0xFFFF) >> 8);
        spi_tx(starting_addr & 0xFF);
        for(i = 0; i < num_bytes; i++){
                buffer[i] = spi_tx(0);
        }
}

/*
 * Writes one byte to memory
 *
 * Note: device address must have been erased for write operation to take place
 */
void flash_write(unsigned long addr, unsigned char value) {

        flash_WREN();

        FLASH_SELECT;
        spi_tx(0x02);
        spi_tx((addr & 0xFFFFFF) >> 16);
        spi_tx((addr & 0xFFFF) >> 8);
        spi_tx(addr & 0xFF);
```

```
        spi_tx(value);
        FLASH_DESELECT;
        wait_flash_busy();
}

/*
 * Waits for any write operation to finish
 *
 * TODO: add detection for infinite wait in case device dies
 *
 */
void wait_flash_busy() {
        while ((flash_status_reg() & 0x01) == 0x01) {
                //_delay_cycles(100);
        }
}

/*
 * Resets whole chip
 *
 */
void flash_reset() {

        flash_WREN();
        FLASH_SELECT;
        spi_tx(0x60);
        FLASH_DESELECT;
        wait_flash_busy();
}

/*
 * Resets SST25VF040B and removes all block write-protection
 *
 */
unsigned char flash_setup() {

        flash_reset();
        flash_WRSR(0);
        return flash_status_reg();
}
```

E.2 BASE STATION CODING

```
#include <stdio.h>
#include <string.h>

#include "spi.h"
#include "cc1101.h"
#include "gpio.h"

// GPIO setup macros. Always use INP_GPIO(x) before using OUT_GPIO(x) or SET_GPIO_ALT(x,y)
#define INP_GPIO(g) *(gpio+((g)/10)) &= ~(7<<(((g)%10)*3))
#define OUT_GPIO(g) *(gpio+((g)/10)) |=  (1<<(((g)%10)*3))
#define SET_GPIO_ALT(g,a) *(gpio+(((g)/10))) |= (((a)<=3?(a)+4:(a)==4?3:2)<<(((g)%10)*3))

#define GPIO_SET *(gpio+7)  // sets   bits which are 1 ignores bits which are 0
#define GPIO_CLR *(gpio+10) // clears bits which are 1 ignores bits which are 0

#define GET_GPIO(g) (*(gpio+13)&(1<<g)) // 0 if LOW, (1<<g) if HIGH

#define GPIO_PULL *(gpio+37) // Pull up/pull down
#define GPIO_PULLCLK0 *(gpio+38) // Pull up/pull down clock

int main(int argc, char *argv[]) {

        setup_io();
        INP_GPIO(CS_GPIO);

        //OUT_GPIO(CS_GPIO);
        //GPIO_SET = 1 << CS_GPIO;

        spi_setup();
        //cc1101_config(2, 0);
        Network_Init(2);

        printf("Finished cc1101 config\n");

        unsigned char rx[64], tx[64];
        int rx_size = sizeof(rx);


        memset(tx, 0, sizeof(tx));
        int i;
        for (i=0; i<sizeof(tx); i++)
                tx[i] = i;


        while(1) {
                //CC1101_strobe(CC_SFRX);
                rx[0] = CC1101_strobe(CC_SRX); //RX mode

                int x;
```

```
                while (!(x = GET_GPIO(13)));

                cc1101_rcv_packet(rx, &rx_size);

                printf("RX %d: 0x", rx_size);
                for (i=0; i < rx_size; i++)
                        printf("%2x ", rx[i]);
                printf("\n");

        }
        printf("Finished sending packet\n");
        spi_close();
}
```

```
int  mem_fd;
void *gpio_map;

// I/O access
volatile unsigned *gpio;


// GPIO setup macros. Always use INP_GPIO(x) before using OUT_GPIO(x) or SET_GPIO_ALT(x,y)
#define INP_GPIO(g) *(gpio+((g)/10)) &= ~(7<<(((g)%10)*3))
#define OUT_GPIO(g) *(gpio+((g)/10)) |=  (1<<(((g)%10)*3))
#define SET_GPIO_ALT(g,a) *(gpio+(((g)/10))) |= (((a)<=3?(a)+4:(a)==4?3:2)<<(((g)%10)*3))

#define GPIO_SET *(gpio+7)  // sets   bits which are 1 ignores bits which are 0
#define GPIO_CLR *(gpio+10) // clears bits which are 1 ignores bits which are 0

#define GET_GPIO(g) (*(gpio+13)&(1<<g)) // 0 if LOW, (1<<g) if HIGH

#define GPIO_PULL *(gpio+37) // Pull up/pull down
#define GPIO_PULLCLK0 *(gpio+38) // Pull up/pull down clock

void setup_io();
```

```
//
//  How to access GPIO registers from C-code on the Raspberry-Pi
//  Example program
//  15-January-2012
//  Dom and Gert
//  Revised: 15-Feb-2013


// Access from ARM Running Linux

#define BCM2708_PERI_BASE        0x20000000
#define GPIO_BASE                (BCM2708_PERI_BASE + 0x200000) /* GPIO controller */

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include "gpio.h"

#define PAGE_SIZE (4*1024)
#define BLOCK_SIZE (4*1024)

void printButton(int g)
{
  if (GET_GPIO(g)) // !=0 <-> bit is 1 <- port is HIGH=3.3V
    printf("Button pressed!\n");
  else // port is LOW=0V
    printf("Button released!\n");
}

//
// Set up a memory regions to access GPIO
//
void setup_io()
{
   /* open /dev/mem */
   if ((mem_fd = open("/dev/mem", O_RDWR|O_SYNC) ) < 0) {
      printf("can't open /dev/mem \n");
      exit(-1);
   }

   /* mmap GPIO */
   gpio_map = mmap(
      NULL,             //Any adddress in our space will do
      BLOCK_SIZE,       //Map length
      PROT_READ|PROT_WRITE,// Enable reading & writting to mapped memory
      MAP_SHARED,       //Shared with other processes
      mem_fd,           //File to map
      GPIO_BASE         //Offset to GPIO peripheral
   );
```

```
    close(mem_fd); //No need to keep mem_fd open after mmap

    if (gpio_map == MAP_FAILED) {
        printf("mmap error %d\n", (int)gpio_map);//errno also set!
        exit(-1);
    }

    // Always use volatile pointer!
    gpio = (volatile unsigned *)gpio_map;


} // setup_io
```

```
/*
 * SPI interface
 *
 *  Created on: Nov 5, 2014
 *      Author: Brandon
 */

#ifndef SPI_H_
#define SPI_H_

//void spi_setup(void (*spi_rx)(char));
void spi_setup();

/* SPI Transmit in Low Power Mode, Interrupt Unsafe */
char spi_tx_lpm_iu(char tx);

/* SPI Transmit in Active Power mode (interrupt safe) */
char spi_tx_am(char tx);

char spi_tx(char tx);

void spi_close();

void spi_burst_tx(char *tx, char *rx, int tx_len);


#endif /* SPI_H_ */
```

```
/*
 * SPI testing utility (using spidev driver)
 *
 * Copyright (c) 2007  MontaVista Software, Inc.
 * Copyright (c) 2007  Anton Vorontsov <avorontsov@ru.mvista.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License.
 *
 * Cross-compile with cross-gcc -I/path/to/cross-kernel/include
 */

/*
 * Source: http://neophob.com/2012/08/raspberry-pi-enable-the-spi-device/
 */

#include <stdint.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>

#include "spi.h"

#define ARRAY_SIZE(a) (sizeof(a) / sizeof((a)[0]))

static void pabort(const char *s)
{
        perror(s);
        abort();
}

static const char *device = "/dev/spidev0.0";
static uint8_t mode = 0;
static uint8_t bits = 8;
static uint32_t speed = 500000;
static uint16_t delay;

static int fd;

static void transfer(int fd, uint8_t *tx, uint8_t *rx, size_t count)
{
        int ret;

        struct spi_ioc_transfer tr = {
                .tx_buf = (unsigned long)tx,
```

```
                .rx_buf = (unsigned long)rx,
                .len = count, //ARRAY_SIZE(tx),
                .delay_usecs = delay,
                .speed_hz = speed,
                .bits_per_word = bits,
        };

        ret = ioctl(fd, SPI_IOC_MESSAGE(1), &tr);
        if (ret < 1)
                pabort("can't send spi message");

}

static void print_usage(const char *prog)
{
        printf("Usage: %s [-DsbdlHOLC3]\n", prog);
        puts("  -D --device   device to use (default /dev/spidev1.1)\n"
             "  -s --speed    max speed (Hz)\n"
             "  -d --delay    delay (usec)\n"
             "  -b --bpw      bits per word \n"
             "  -l --loop     loopback\n"
             "  -H --cpha     clock phase\n"
             "  -O --cpol     clock polarity\n"
             "  -L --lsb      least significant bit first\n"
             "  -C --cs-high  chip select active high\n"
             "  -3 --3wire    SI/SO signals shared\n");
        exit(1);
}

static void parse_opts(int argc, char *argv[])
{
        while (1) {
                static const struct option lopts[] = {
                        { "device",  1, 0, 'D' },
                        { "speed",   1, 0, 's' },
                        { "delay",   1, 0, 'd' },
                        { "bpw",     1, 0, 'b' },
                        { "loop",    0, 0, 'l' },
                        { "cpha",    0, 0, 'H' },
                        { "cpol",    0, 0, 'O' },
                        { "lsb",     0, 0, 'L' },
                        { "cs-high", 0, 0, 'C' },
                        { "3wire",   0, 0, '3' },
                        { "no-cs",   0, 0, 'N' },
                        { "ready",   0, 0, 'R' },
                        { NULL, 0, 0, 0 },
                };
                int c;

                c = getopt_long(argc, argv, "D:s:d:b:lHOLC3NR", lopts, NULL);
```

```
                if (c == -1)
                        break;

//https://www.kernel.org/doc/Documentation/spi/spi-summary
                switch (c) {
                case 'D':
                        device = optarg;
                        break;
                case 's':
                        speed = atoi(optarg);
                        break;
                case 'd':
                        delay = atoi(optarg);
                        break;
                case 'b':
                        bits = atoi(optarg);
                        break;
                case 'l':
                        mode |= SPI_LOOP;
                        break;
                case 'H':
                        mode |= SPI_CPHA; //Clock phase, 1=sample on trailing, 0=sample on
                        break;
                case 'O':
                        mode |= SPI_CPOL; //Clock polarity, 1=Clock starts high, 0=Clock st
                        break;
                case 'L':
                        mode |= SPI_LSB_FIRST;
                        break;
                case 'C':
                        mode |= SPI_CS_HIGH;
                        break;
                case '3':
                        mode |= SPI_3WIRE;
                        break;
                case 'N':
                        mode |= SPI_NO_CS;
                        break;
                case 'R':
                        mode |= SPI_READY;
                        break;
                default:
                        print_usage(argv[0]);
                        break;
                }
        }
}

void spi_setup()
{
        int ret = 0;
```

```
        mode = 0;

        fd = open(device, O_RDWR);
        if (fd < 0)
                pabort("can't open device");

        /*
         * spi mode
         */
        ret = ioctl(fd, SPI_IOC_WR_MODE, &mode);
        if (ret == -1)
                pabort("can't set spi mode");

        ret = ioctl(fd, SPI_IOC_RD_MODE, &mode);
        if (ret == -1)
                pabort("can't get spi mode");

        /*
         * bits per word
         */
        ret = ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits);
        if (ret == -1)
                pabort("can't set bits per word");

        ret = ioctl(fd, SPI_IOC_RD_BITS_PER_WORD, &bits);
        if (ret == -1)
                pabort("can't get bits per word");

        /*
         * max speed hz
         */
        ret = ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
        if (ret == -1)
                pabort("can't set max speed hz");

        ret = ioctl(fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
        if (ret == -1)
                pabort("can't get max speed hz");

        printf("spi mode: %d\n", mode);
        printf("bits per word: %d\n", bits);
        printf("max speed: %d Hz (%d KHz)\n", speed, speed/1000);

}


char spi_tx(char tx) {
        char rx;
        transfer(fd, &tx, &rx, 1);
        return rx;
```

```
}

void spi_burst_tx(char *tx, char *rx, int tx_len) {
        transfer(fd, tx, rx, tx_len);
}

void spi_close() {
        close(fd);
}
```

References

[1] R.A. Dalke, C.L. Holloway, P. McKenna, M. Johansson, and A.S. Ali. Effects of reinforced concrete structures on rf communications. *Electromagnetic Compatibility, IEEE Transactions on*, 42(4):486–496, Nov 2000.

[2] Shan Jiang. Optimum wireless power transmission for sensors embedded in concrete. Master's thesis, Florida International University, nov 2011.

[3] O. Jonah and S.V. Georgakopoulos. Efficient wireless powering of sensors embedded in concrete via magnetic resonance. In *Antennas and Propagation (APSURSI), 2011 IEEE International Symposium on*, pages 1425–1428, July 2011.

[4] William C. Stone. Electromagnetic signal attenuation in construction materials. *NIST Construction Automation Program Report No. 3*, 1997.

[5] Clayborne D. Taylor, Samuel J. Gutierrez, Steven L. Langdon, Kenneth L. Murphy, and III Walton, William A. Measurement of rf propagation into concrete structures over the frequency range 100 mhz to 3 ghz. In JeffreyH. Reed, TheodoreS. Rappaport, and BrianD. Woerner, editors, *Wireless Personal Communications*, volume 377 of *The Springer International Series in Engineering and Computer Science*, pages 131–144. Springer US, 1997.

List of Figures

1 System block diagram . . . 4
2 Wireless Power Transmission . . . 6
3 Royer Oscillator Circuit . . . 6
4 Receiver . . . 7
5 Node encased in concrete after recovery . . . 9
6 Top view of design for both charging and non-charging enclosures . . . 11
7 Front view of design for non-charging enclosure . . . 11
8 Front view of design for charging enclosure . . . 11
9 Charging Chip Circuitry . . . 12
10 Non-Charging PCB . . . 13
11 Charging PCB . . . 13
12 Multisim schematic for the charging PCB. . . . 13
13 Parts List . . . 14
14 Parts List for Non-Charging Enclosure . . . 15
15 Parts List for Charging Enclosure . . . 15
16 Transmitter Part List . . . 15
17 Packet success rate for different output powers . . . 16
18 Load Analysis . . . 17
19 Efficiency . . . 17
20 Battery life expectancy and current draw . . . 18
21 Nylon mesh covering sensor opening . . . 20
22 Locations of sensor openings with 90 degree rotation . . . 21
23 Air gap between enclosure walls . . . 22
24 Concept sketch of an alternate way to charge and retrieve data . . . 23
25 Individual Contributed Hours . . . 25
26 Individual Contributed Hours and Total Time . . . 25