OTTER GPS IMPLANT

By

Andy Beugelsdijk

Nicholas Gruebnau

Sugato Ray

Final Report for ECE 445, Senior Design, Spring 2012

TA: Justine Fortier

May 2 2012

Project No. 19

`

Abstract

Our group's project this semester was to design a sub-dermal GPS implant for use under a North American River Otter's skin to allow local biologists to track their locational data as they continued about their regular behavior. While there are a number of special devices available to biologists for use to track animals, none of them are specialized enough to cater to what is required when it comes to tracking otters. Our device uses small parts like the Arduino Pro Mini and a helical antenna GPS module to ensure the smallest possible size and weight.

Our project's circuitry ended up being completely functional with the exception of a damaged trace on the PCB. Nicholas has plans to continue work on the project to refine the PCB and produce a field ready module next semester.

`

Contents

1. Introduction 1

1.1 Specifications 1

1.1.1 Tracking Unit 1

1.1.2 Base Station 1

2. Design 2

2.1 Block Diagram 2

2.1.1 Description of Block Diagram 3

2.2 Choice of Major Components 4

2.3 Equations and Calculations 5

2.4 Schematic 7

2.5 Programming 7

2.6 Design Alternatives 8

3. Design Verification 8

3.1 Tracking Unit 8

3.1.1 GPS Chip 8

3.1.2 Arduino Mini Pro 9

3.1.3 Battery 9

3.2 Base Station 9

3.2.1 Arduino Uno 9

3.2.2 Data Logger 9

3.2.3 Linx LT Series Pair 9

4. Costs 10

4.1 Parts 10

4.2 Labor 11

5. Conclusion 11

5.1 Accomplishments 11

5.2 Uncertainties 11

5.3 Ethical considerations 11

5.4 Future work 12

References 13

Appendix A Requirement and Verification Table 14

Appendix B GPS Implant PCB Design 17

Appendix C GPS Satellite Fix 17

Appendix D Implant Arduino Code 18

Appendix E Base Station Arduino Code 24

`

1. Introduction

Wildlife biologists have recently started using advanced technology to monitor the behavior of animals in their natural habitat. In particular, the return of the North American river otter population in Illinois has piqued the interest of many local biologists. The otter was nearly considered extinct in Illinois around the 1800s due to human interactions. After a recent reintroduction of the otter to Illinois the animal seems to be doing well and populations are increasing. Scientists are now seeking the use of technology in order to learn why they are thriving and what effects they will have on the environment.

Scientists frequently use collar mounted GPS tracking systems in order to record the traveling habits of larger mammals. While these have proven useful for larger animals, animals such as the otter pose an interesting problem in that their heads are smaller than their necks. Any sort of collared device would simply slip off of the animal due to their active lifestyle. Similarly, harnesses or other exterior devices run the risk of getting caught underwater and potentially suffocating the animal. Biologists studying the otter are in need of sub-dermal tracking methods in order to safely monitor the animal without altering its usual behavior.

The goal of our group was to develop a sub-dermal GPS implant for use with the North American River Otter, per our biologist contact Samantha Carpenter's request. The project was designed as small possible and as power efficient as possible, with an emphasis on longevity of the device and the ease of retrieval of data. Taking this into account, our group decided that, along with the implant, we would create a "station" which would be located on a strip of land that the otter's frequented (called a leyline). The station would be able to wirelessly collect the data and write it to a USB device which the biologist could collect when they came to visit the leyline for research performed roughly twice a week.

1.1 Specifications

Our project was split into two parts, the GPS implant which was to be inserted under the otter's skin, and the station which was to be placed in the center of the otter's leyline.

1.1.1 Tracking Unit

The GPS Implant consists of a GPS chip which communicates with satellites and triangulates its current position by pinging a minimum of four satellites. This data is passed along to a microcontroller which saves it until the unit is within range of the base station. Finally, once the otter is in range of the base station, the microcontroller sends the data to the wireless transceiver which uses an antenna to transmit the data to another transceiver located at the base station.

1.1.2 Base Station

The Base Station consists of a large microcontroller which receives the location data from its connected transceiver. It then sends this information to a data logger which writes the latitudes, longitudes, altitudes, times, and dates to the USB flash drive.

`

2. Design

The main focus of our design procedure was the implantable tracking unit, and it was a fairly complicated one. Two things were of utmost importance- the physical size of the implantable device, and its power consumption. The implanted device is meant to be surgically implanted into an otter at a field location, and hence the surgery has to as non-intrusive as possible. To accommodate this, the device has to be extremely small. Moreover, the device is in no way meant to cause the animal to divert in any manner from its natural habitual behaviors, as this would completely defeat its major purpose of studying the behavior of otters. Concurrently, the tracking unit has to consume very little power as it would be very illogical and unethical to keep performing surgeries on an animal to replace the batteries on the device. Keeping these two factors in mind, research was conducted on similar commercially available products. Following that, a block diagram was made and a conclusion was drawn on the choice of parts for the implantable unit. Then, parts for the base station were chosen to complement the components of the implant. The following subsections contain detailed of the entire design procedure.

2.1 Block Diagram

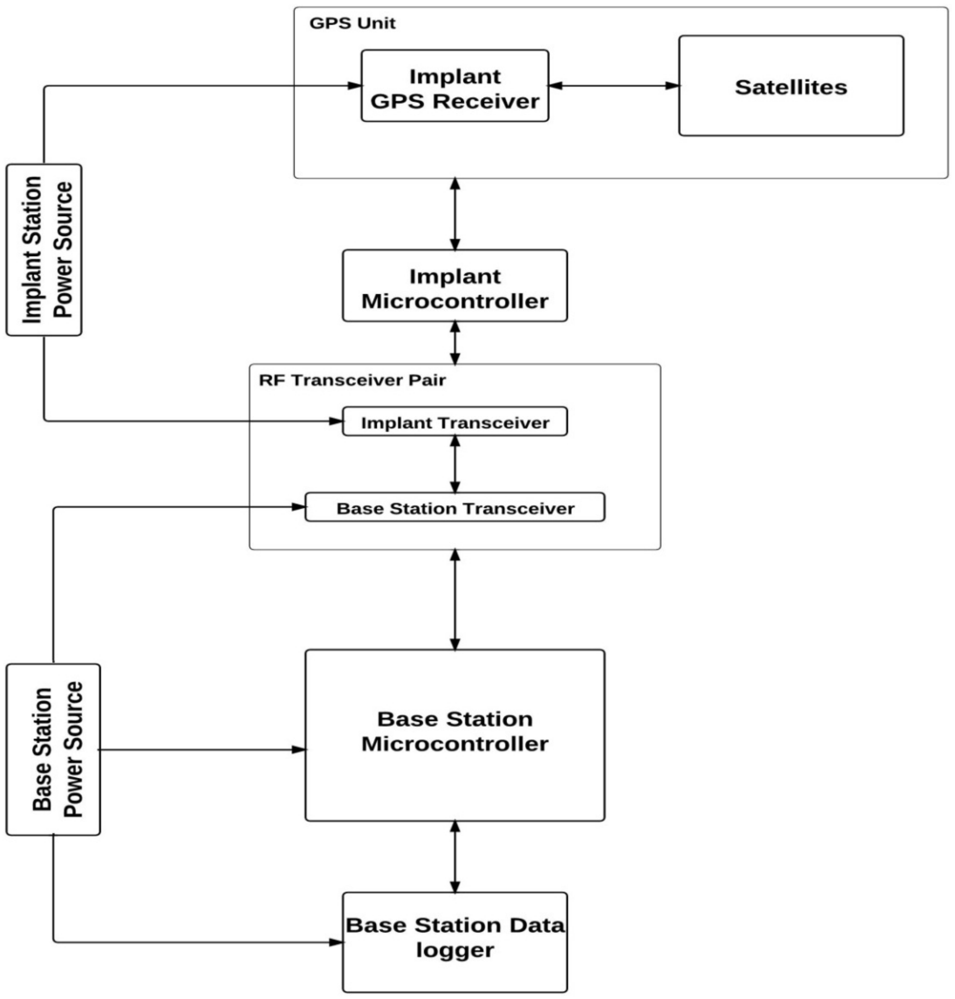


Figure 1: Project Block Diagram

`

2.1.1 Description of Block Diagram

Satellites:

There are a minimum of 24 GPS satellites orbiting the globe at all times. These are utilized by the GPS implant to obtain positional information on the otters as well the UTC time at which these positional data points are obtained.

Implant GPS Receiver:

The GPS unit consists of a programmable GPS chip with onboard storage and read/write capability. It will receive positional coordinates and timestamps from the available satellites at periodic intervals. Then, the information will be conveyed implant microcontroller.

Implant Microcontroller:

The implant microcontroller parses the GPS data and stores the relevant information (location and timestamps) onto its onboard memory. The microcontroller also receives signals from the transceiver in order to determine if it is within a transmittable range from the base station. If it is, the microcontroller will transmit the data to the base station via the transceiver and wipe its memory for future use.

RF transceiver pair:

The implant and base station transceivers act as intermediaries between the implant and base station microcontrollers. They, along with the antennas attached to them, govern the wireless communication between the base station and implant, and the maximum transmission range.

Base Microcontroller and Data Logger:

The microcontroller controls the RF transceiver to extract data from the implant when the otter is within an acceptable range, and finally stores it in the data logger. The data logger is basically a USB flash drive connected to a USB-to-RS232 converter which is connected to the base station microcontroller. Software simulated serial communication is used to write relevant data to the USB flash drive. The microcontroller will also send out a signal via the transceiver signaling that data transfer is complete.

Implant Power Source:

The implant power source is a 9V, 1200mAh battery as this would ensure that the device remains fully functional for at least 2 weeks. IC voltage regulators are used to regulate the voltage in order to achieve the required input voltage of 3.3V. Biologists are interested in accurate location data that can only be provided by an actively powered unit. They assured us that they have no concerns with the inevitability of the unit requiring a battery replacement.

Base Station Power Source:

For the base station, there is no strict requirement when it comes to the power source, as long as the battery life of the base station is longer than that of the implantable unit. To achieve this, a logical option would be to use a two 3.7V, 6000mAH Lithium Polymer batteries, and regulate the voltage through IC regulators to achieve an input voltage of 5V.

`

2.2 Choice of Major Components

Component	**Part Chosen**	**Description and Reasoning**
Implant GPS Receiver	µblox D2523T	• Has helical antenna which reduces its dependence on orientation as the RF waves would be circularly polarized • Low power consumption • Highly programmable • Serially communicates with microcontroller
Implant Microcontroller	Arduino Pro Mini	• Very small and has numerous software libraries to work with • 3.3V TTL level making it compatible with GPS and Transceiver • Can power GPS and transceiver
RF transceiver pair	Linx TRM-433-LT (433MHz)	• Meant to be used in pairs and designed to communicate with each other serially at various baud rates • Low power consumption
Antenna for RF transceivers	"The Splatch" - Grounded Line ¼-Wave Monopole Antenna - 433 MHz	• Matched to the frequency and output line impedance (50 Ω) of the transceivers • PCB embeddable and small • Omnidirectional • Identical antennas were chosen for both transceivers to avoid polarization mismatch.
Base Microcontroller	Arduino Uno SMD Edition	• Large flash memory • Identical programming environment as implant microcontroller
Data Logger	Parallax Datalogger	• Arduino compatible • Large set of firmware commands

`

2.3 Equations and Calculations

The main calculations that had to be performed for this project were the power calculations as the power budget for the implantable device is fairly small. These calculations for the implantable device were done to determine how long the device should run at its full capacity, and also to determine how long the device would run once the proper scheduling mechanism was programmed into it. The battery used was a 9V, 1200mAH Lithium battery. All our devices ran at an operating voltage of 3.3V. Their procedures and results are shown next. These results were used to engineer our verification procedures for the battery life.

Full Capacity Operation:

First fifteen minutes for startup during satellite signal acquisition,

$$\left.\begin{aligned} I_{GPS,acquisition} &= 47mA \\ I_{GPS,tracking} &= 11mA \\ I_{microcontroller} &= 40mA \\ I_{Transceiver} &= 7mA \end{aligned}\right\} \longrightarrow \begin{aligned} I_{Total,1} &= 47 + 11 + 40 + 7 \\ I_{Total,1} &= 105\ mA \end{aligned}$$

Subsequent operation,

$$\left.\begin{aligned} I_{GPS,tracking} &= 11mA \\ I_{microcontroller} &= 40mA \\ I_{Transceiver} &= 7mA \end{aligned}\right\} \longrightarrow \begin{aligned} I_{Total,2} &= 11 + 40 + 7 \\ I_{Total,2} &= 58\ mA \end{aligned}$$

Let *t* be the operation time of the devices running at $V_{in}=3.3V$ after the initial 15 minutes,

$$\mathbf{0.25(I_{Total,1}) + I_{Total,2} \times t = \mathit{MilliAmphour\ capacity\ of\ battery} @ 3.3V}$$

$$0.25(105mAh) + 58mA \times t\ hours = 1200mAh \times \frac{9}{3.3}$$

$$t = \frac{1200 \times \frac{9}{3.3} - 0.25(105)}{58}\ hours$$

$$\therefore \boldsymbol{t = 56\ hours}$$

So, based on the above calculations, the battery should last about 56 hours when the implant is being run at full capacity. However, this number should drastically increase once the scheduling mechanism had been programmed into the implantable unit. Firstly, under clear weather conditions, the GPS

`

receiver is able to acquire a signal from the satellites within 90 seconds to 3 minutes depending on how attenuated the signal is due to the surroundings. Furthermore, the entire implantable unit “sleeps” for most of the day, and runs for a total of only 60 minutes a day. Considering these factors, the following set of calculations were done, while keeping the satellite signal acquisition time at 15 minutes to account for bad weather.

During 60 minutes of full capacity operation during the day,

$$\left.\begin{array}{ll} I_{GPS,acquisition} & = 47mA \\ I_{GPS,tracking} & = 11mA \\ I_{microcontroller} & = 40mA \\ I_{Transceiver} & = 7mA \end{array}\right\} \longrightarrow \begin{array}{l} I_{Total,1} = 47 + 11 + 40 + 7 \\ I_{Total,1} = 105\ mA \\ Total\ milliamp\ hours\ drawn = 105mA \times 1hour \\ \qquad = \boldsymbol{105mAh\ daily} \end{array}$$

For the rest for the 23 hours of the day while device is in sleep mode,

$$\left.\begin{array}{ll} I_{GPS,tracking} & = 0mA \\ I_{microcontroller} & = 0.33mA \\ I_{Transceiver} & = 0mA \end{array}\right\} \longrightarrow \begin{array}{l} I_{Total,2} = 0.33\ mA \\ Total\ milliamp\ hours\ drawn = 0.33mA \times 23hours \\ \qquad = \boldsymbol{7.59mAh\ daily} \end{array}$$

Let d be the number of days the device should operate (battery life of device),

$$(105mAh + 7.59mAh)d = 1200 \times \frac{9}{3.3}$$

$$d = \frac{1200 \times \frac{9}{3.3}}{105 + 7.59}\ hours$$

$$d = 29.06\ days \ \ \rightarrow \boldsymbol{d \approx 29\ days\ of\ operation}$$

Therefore, from the calculations shown above, it was deduced that the device would run for about 4 weeks under moderate weather conditions. When it comes to the 60 minutes it operates at full capacity a day, the device should be able to acquire at least 4 data points given the skeptically estimated satellite signal acquisition time of 15 minutes. Realistically, higher expectations can be set as the signal acquisition time is usually much shorter as hostile weather conditions are not that frequent.

`

2.4 Schematic

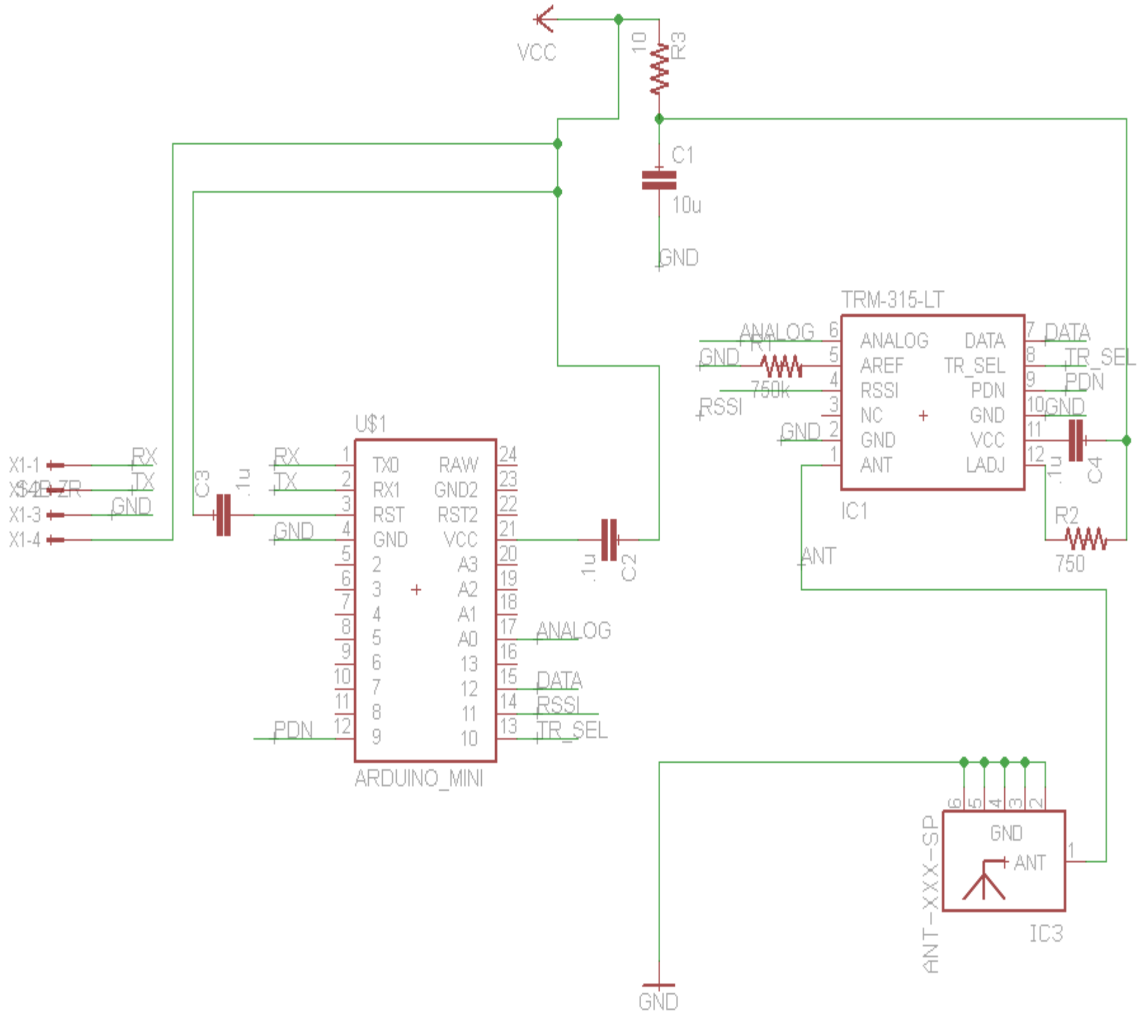


Figure 2: Schematic

2.5 Programming

The design of the code began with first listing the necessary tasks that needed to be accomplished by the overall system. For the Implant, it needs to successfully obtain GPS data several times a day and store this data temporarily until it can be transferred to the Base Station. For the Base Station, it needs to take the wirelessly transferred data from the Implant and store it onto a USB drive. For both the Implant and the Base Station, they both need to have a way of listening for and detecting each other. The code for the Implant ended up having three main sections: GPS parsing, time monitoring, and Base Station communication. Similarly, the code for the Base Station was also made up of three main functions: detection signal broadcasting, Implant communication, and data transfer to the USB stick. The final codes for both the Implant and Base Station can be found in Appendix D and Appendix E, respectively.

`

2.6 Design Alternatives

There were a few other things that our group was considering regarding the design of our project. Initially, we thought of using a long, high gain, linearly polarized antenna for the GPS to improve our signal strength and have it stick out of the otter. However, the only problem with that was the orientation of the antenna would greatly affect the functionality of the device. So, this idea was discarded.

Another alternative that was being considered was the use of two 3.7V, 2000mAh Lithium Polymer batteries instead of a single 9V, 1200mAh battery for the implant. These Lithium Polymer batteries are much thinner, and lighter than the 9V, 1200mAh battery. However, they are about 1.5 times wider. Therefore, we went with the 9V battery instead.

We were also thinking of having a dedicated receiver and a transmitter on both the implantable unit and the base station so that both devices could transmit and receive data simultaneously. However, due to our small power budget regarding the implant, we had to discard this idea.

3. Design Verification

Our project met all of the verifications that the group had set for itself. While the functionality of the Linx LT Series Pair was not able to be demonstrated, the parts successfully transmitted the data in a lab setting.

3.1 Tracking Unit

The tracking unit was successful in its acquisition of data and transferring of said data to the base station.

3.1.1 GPS Chip

The GPS chip successfully acquired data that was accurate to our true location. This was verified using the Google location API through Google Earth against a GPS fix given by the chip. As shown in the table below, the GPS receiver's data was more than 99% accurate.

Test Vs. Google Location API	**Implant GPS Unit [From 169EL]**	**Actual [From Google Location API]**	**Percentage Error [%]**
Latitude [^{o}]	-88.227829	-88.27945	0.05847454
Longitude [^{o}]	40.110867	40.110926	0.00014709
Time [UTC]	20:03:38	20:03:38	0
Elevation [m]	213.00	212.00	-0.47169811
Date [M/D/Y]	3 / 27 / 2012	3 / 27 / 2012	0

`

3.1.2 Arduino Mini Pro

The Mini Pro successfully stored data on to its onboard memory and once the data was sent it deleted the old data to make room for the new. This was verified by sending data in one of its RxI pins and then trying to access it through its TxO pin. The memory was then checked through Arduino's serial monitor; the memory locations were deleted once the data was sent.

3.1.3 Battery

By stress testing the unit under different conditions (bad weather, through skin, various media) we were able to get an idea of the power the implantable unit would consume. When we performed the stress test, our 9V, 1200mAh battery lasted for 48 hours while running at full capacity under moderate weather conditions.

3.2 Base Station

The Base Station was successful in collected the data from the implant and writing the data to a data logger.

3.2.1 Arduino Uno

The Arduino Uno successfully parsed the data from the station transceiver to the data logger so that it could be written.

3.2.2 Data Logger

The Data Logger successfully wrote the data to a USB. The group then opened the file on the USB and ensured that it had been written in a manner that was clear and concise.

3.2.3 Linx LT Series Pair

The Linx LT Series Pair was able to transmit messages back and forth in the lab over the small distances (5ft) that we required. They were not shown, however, on the PCB. Below is a screenshot captured on the oscilloscope of the output of one of the transceivers that was receiving a signal from the other a small distance away. Signal strength pin [RSSI] was probed and the measured voltage was shown to correspond to a signal strength of -5dB, as per our verifications.

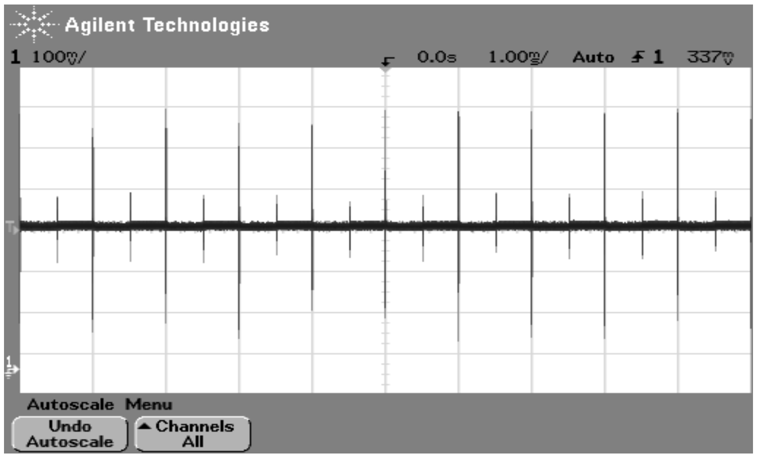


Figure 3: Linx Transceiver RSSI

`

4. Costs

4.1 Parts

Part	Manufacturer	Retail Cost ($)	Bulk Purchase Cost ($)	# of Units Bought	Actual Cost ($)
50 Channel D2523T Helical GPS Receiver	ADH Technology Co. Ltd.	79.95/unit	71.96/unit for 10-99	1	79.95
Linx LT Series 433MHz RF Transceiver Module	Linx Technologies Inc.	15.47/unit	13.43/unit for 100	2	30.94
Linx 433MHz SP Series “The Splatch” Antenna	Linx Technologies Inc.	2.41/unit	2.03/unit for 100	2	4.82
Arduino Pro Mini 328-3.3V/8MHz	Arduino	18.95/unit	17.06/unit for 10-99	1	18.95
FTDI Basic Breakout-3.3V	Future Technology Devices International Ltd.	14.95/unit	13.46/unit for 10-99	1	14.95
FTDI Cable 5V VCC-3.3V I/O	Future Technology Devices International Ltd.	17.95/unit	16.16/unit for 10-99	1	17.95
Arduino Uno SMD	Arduino	29.95/unit	26.96/unit for 10-99	1	29.95
Memory Stick Datalogger	Parallax Inc.	39.99/unit	35.99/unit for 10	1	39.99
Ultralife 9V Lithium Battery-1200mAh	Ultralife	7.95/unit	7.50/unit for 12-71	1	7.95
Polymer Lithium Ion Battery-6Ah	Union Battery Corp.	39.95/unit	35.96/unit for 10-99	1	39.95
USB 2.0 A Male to B Male Cable (1m)	Cables to Go	7.99/unit	-------	1	7.99
USB 2.0 A Male to Mini-B Male Cable (1m)	Cables to Go	11.99/unit	-------	1	11.99
				Total Cost	**305.38**

`

4.2 Labor

Engineers	Rate ($)	Hours/Week	Total # of Weeks	Multiplier	Total ($)
Sugato Ray	40/hr	10	13	2.5	13,000
Nick Gruebnau	40/hr	10	13	2.5	13,000
Andrew Beugelsdijk	40/hr	10	13	2.5	13,000
				Total Cost	**39,000**

5. Conclusion

Our group was very pleased with the outcome of our project and the reception with which it was received. The demonstration proved that the circuitry was completely functional baring the damaged wireless transceiver. Having proven the transceiver's functionality prior to the demonstration, however, the group is confident that the parts can be replaced with ease and be installed on a new PCB for a field-ready project.

5.1 Accomplishments

Our project succeeded in our goal of obtaining accurate location information while providing long battery life. Conservative estimates of the Tracking System's battery life were roughly twenty one days, or a week longer than we had initially intended. While it is possible that this lifetime may change once the device is properly housed, we believe that the time span provided will be sufficient for the biologists needs.

The device was also capable of getting location data (time, latitude, longitude) accurately; to within less than one percent error of that provided by Google Maps when provided with an address.
Finally, and perhaps most importantly, the biologists were satisfied with the size of the device which was shown both as its PCB footprint and with all of the parts attached. Given proper functionality, good life span, and acceptable size, our group believes we accomplished our major goals for this project.

5.2 Uncertainties

Some uncertainties do remain with the housing for the implant and base station. At the time of this writing, we are considering using a silicon based coating so as to ensure an ergonomic design for the implant. Any weather proof casing for the base station will suffice. The entire system's functionality will need to be retested once proper housing has been found for both units.

5.3 Ethical considerations

Given that this project is to be inserted under the skin of a living animal, the group had many ethical issues to consider. Comfort of the animal was a primary concern, and the implant portion was designed to be as small as possible so as not to cause discomfort during the otter's daily activities. It was also constructed to weight no more than 3 pounds, or 10% of the otter's total body weight.

`

5.4 Future work

One group member, Nicholas Gruebnau, will be taking an independent study next semester to finalize the PCB design, add the transceivers to the circuit, and find suitable housing so as to create a field ready product that the biologists can begin using.

New PCB designs are being constructed to further reduce the size of the implantable device. One of our Group members, Sugato Ray, is currently corresponding with his old employers about helping the group make new PCBs on flexible substrates to make it easier for the biologists to surgically implant this device.

Furthermore, new PCB designs for the base station are also being made to reduce the noise floor that the Transceiver would have to deal with in order to increase the fidelity of received signals. This would also help increase the range at which the base station as implantable unit can communicate with each other as it would make it possible for the Transceiver to accurately decipher weaker signals.

Our Group is also looking into a new device to be added onto the base station, name the Linx HP series RF module. This device has the capability of working on different frequencies. This will allow one to differentiate one otter from another through the use of frequency signatures.

`

References

[1] Linx Technologies LT Series, web page. Available at: http://www.linxtechnologies.com/products/rf-modules/lt-series-transceiver-modules/. Accessed April 2012.

[2] Arduino CC, web page. Available at : http://arduino.cc/en/Tutorial/HomePage. Accessed April 2012.

[3] Parallax, web page. Available at : http://www.parallax.com/tabid/768/ProductID/434/Default.aspx. Accessed April 2012.

[4] Misra, Pratap, and Per Enge. *Global Positioning System: Signals, Measurements, and Performance.* Lincoln: Ganga-Jamuna Press, 2006. Print.

`

Appendix A Requirement and Verification Table

Requirement	Verification	Verification Status
1.GPS Unit -Acquires signals from 4 satellites to gain positional information. a. Gets accurate information of location. Error : location <10m; timestamp within < 1s of time taken. b. Must be able to run on power saving mode and ping satellites defined by user programing. c. Must be able to communicate with microcontroller.	1. The GPS unit is connected to the Arduino Mini. It receives instructions from the unit as well as parses data to the microcontroller. To test it, we can use a user interface provided by the manufacturer and monitor its outputs using a bread board. a. GPS unit will be turned on, connected to bread board and then to PC. Data will be obtained using NMEA parsing software. This will ensure accuracy and precision of data. b. User defined program will be uploaded using ublox-6 software to determine ping intervals and minimize power consumption. c. Serial input/output port (Tx, Rx) will be connected to Arduino Mini to test communication between then and data transfer rate.	Verified
2. Arduino Mini (Implant Microcontroller) -Arduino mini will receive location data and time stamp from GPS and store it onto onboard memory. Then it will convey the information to RF transceiver when it is in range of the base station. a. Has to parse NHEA data received and store only relevant information, namely timestamp and location. b. Has to receive signal from Linx module confirming range and pass data to it for transmission. Then it	2. The Arduino Mini will be attached to a bread board and connected to an Arduino Uno (already owned). The Uno is recommended by Arduino as a solid debugging/programing tool as it can connect to the Mini and use its USB port to connect to a computer for a user interface. a. Will use bread board and upload NMEA parsing program and collect test data to see if accurate location and time information is stored onto memory without using too much memory space. b. Connect Linx module to microcontroller and see if it can recognize its input signal	Verified

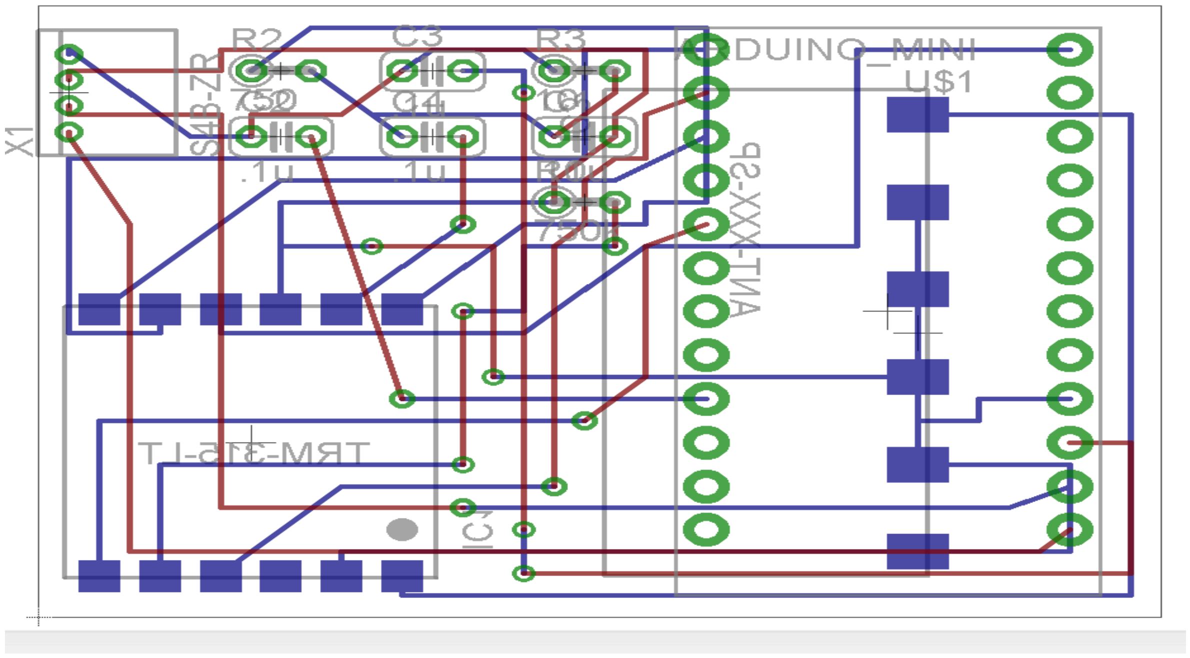
must wipe the transmitted data from memory. c. Has to run in power save mode to minimize power consumption.	to confirm transmittable range and then send out the information back to it at a matched data transfer rate. We will also ensure that the data that has been transferred will be wiped from memory once stop signal is received from RF transceiver. c. Will upload program that puts the microcontroller to sleep mode and verify that it turns on only when the GPS module is about to ping the satellites by making use of the start bit sent out by the GPS module.	
3. RF Transceiver Pair -The transceiver pair must be able to communicate with each other and transfer data accurately. The Linx patch and standard monopole antennas must be able to communicate within a reasonable range of at approximately 5ft so that we can get the data from the otter. a. Range of transmission should be about 5ft. b. Sends data between each pair and convey the data received to their respective microcontrollers.	3. The transceivers can be wired to the Arduino Mini/Uno on a bread board and the inputs/outputs monitored to ensure it will send or receive the correct data. We also will have a manufacturer made evaluation kit to ensure our antenna's transmission range will be correct. a. Put the transceivers 5ft apart and adjust the output power level of the transceiver using the LADJ pin to see if a minimum of -5dBm signal is obtained. b. Will send arbitrary data trough microcontroller and see if it can be received and transferred to the microcontroller.	Verified in lab, but not successfully implemented onto PCB
4. Station Microcontroller (Arduino Uno) -Must receive data from implant and store send it to the Data Logger protocol to be written to the USB drive. a. Must be able to communicate with transceiver in order to receive information and change	4. The microcontroller can be powered and analyzed using a simple bread board. It has a USB port so that it can be easily programmed using manufacturer provided user interface. a. Will serially upload information to Linx and see if the microcontroller can receive the data. We will also monitor the Linx transceiver and make sure that the Uno is	Verified

`

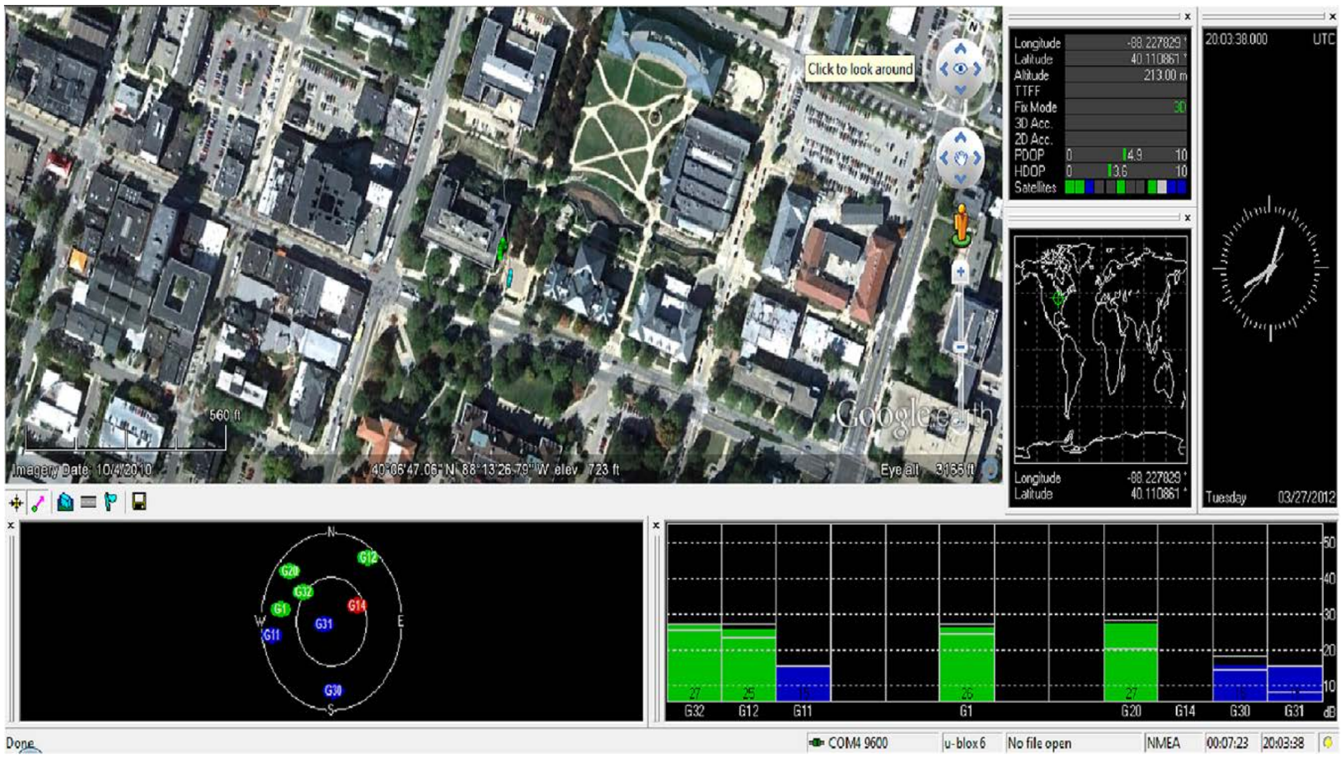
transmitting/receiving mode. b. Must be able to send data to Data Logger block.	correct managing the transceiver's send/receive protocols. b. Will send data to Data Logger and ensure that the correct data appears on that end.	
5. Data Logger -Must receive data from Arduino Uno and write data to the USB. a. Will write data the USB in an easy to understand format for use by the biologists.	5. The Data Logger block is within the microcontroller. It is programmed the same way the Uno will be. a. Will send arbitrary data to the microcontroller's writing protocol and ensure that it records the data properly.	Verified
6. GPS Power Source - Implant must record location data at a high enough frequency that it will provide an accurate representation of the traveling habits of the otter. At the same time, the more often the chip pings satellites, the faster we will drain the battery. **a.** Must provide at least 2 locations per day and will last approximately 2 weeks.	6. Unit will be "stress tested" using different satellite pinging rates. We can then use voltmeter to see how long the required voltage can be maintained. a. Will test battery life starting with at least 2 data collections each day, and slowly increase until we find a reasonable balance.	Verified

`

Appendix B GPS Implant PCB Design



Appendix C GPS Satellite Fix



`

Appendix D Implant Arduino Code

```
#include <SoftwareSerial.h>

//GPS PARSING VARIABLES
int rxPin = 1; //gps serial rx pin
int txPin = 0; //gps serial tx pin
int powerGPS = 4; //gps power pin //CHECK IMPLANT PCB AND FIX ACCORDINGLY
int byteGPS = -1; //used for storing an incoming gps byte
char NMEA_code[7] = "$GPRMC"; //used for specifying gps output to be parsed
boolean code_match = true; //will determine if NMEA_code is matched
boolean ping_successful = false; //will determine when gps data has successfully been obtained
char buffer[300]; //used for storing incoming gps bytes
int buffer_length = 0; //gives the length of buffer being used
int indices[13]; //used for storing the indices of the buffer that correspond to a ',' or '*' character
int indices_length = 0; //gives the length of indices being used

//TIME MONITORING VARIABLES
boolean StartUp = true; //shows that the implant has just been started (or restarted)
int theTime = 0; //keeps track of the hr of the day in min (1 hr = 1,440 min)
unsigned long duration = 0; //keeps track of the # of min passed since theTime was last updated
unsigned long prev_millis = 0; //stores the millisecond count from when theTime was last updated
unsigned long curr_millis = 0; //stores the current millisecond count while updating theTime

//TRANSCEIVER INTERFACE & BASE STATION COMMUNICATION VARIABLES
SoftwareSerial transSerial_R(12, 13); //transceiver serial receive, rx is pin 12 (pin 13 is dummy tx)
SoftwareSerial transSerial_T(13, 12); //transceiver serial transmit, tx is pin 12 (pin 13 is dummy rx)
int TR_Pin = 10; //transceiver transmit/receive select pin (HIGH = transmit, LOW = receive)
int PDN_Pin = 9; //transceiver power down pin (HIGH = power on, LOW = power down) this pin should always be HIGH
boolean transmit_successful = false;
int transmit_time = -1;
char byteIn = 0; //used for storing an incoming transceiver byte
char base_signal = 63; //detection byte that the base station is broadcasting (decimal 63 equals a '?' character)
char implant_response = 33; //response byte that the base station listens for (decimal 33 equals a '!' character)

//DATA STORAGE & MESSAGE STRUCTURE VARIABLES
const int max_length = 100;
char lat[max_length]; //buffer for storing latitude values
char lon[max_length]; //buffer for storing longitude values
char time[max_length]; //buffer for storing time-stamp values
char date[max_length]; //buffer for storing date-stamp values
int lat_length = 0; //gives the length of lat being used
int lon_length = 0; //gives the length of lon being used
int time_length = 0; //gives the length of time being used
int date_length = 0; //gives the length of date being used
char byteStart = 1; //the start byte is a "start of heading" character
char byteEnd = 10; //the end byte is a "new line" character
char byteStop = 4; //the stop byte is a "end of transmission" character
char end_of_instance = 13; //used to separate instances of gps data (decimal 13 = 'carriage return')
//MESSAGE FORMAT: byteStart, (latitudes, byteEnd, longitudes, byteEnd, times, byteEnd, dates, byteEnd, byteStop)x3, byteStart



void setup(){
 pinMode(rxPin, INPUT);
 pinMode(txPin, OUTPUT);
 pinMode(TR_Pin, OUTPUT);
 pinMode(PDN_Pin, OUTPUT);
 digitalWrite(PDN_Pin, HIGH); //power up transceiver
 digitalWrite(TR_Pin, LOW); //set transceiver to receive
 pinMode(12, INPUT); //set transSerial_R rx pin as digital input
 Serial.begin(9600);
 transSerial_R.begin(9600);
 transSerial_T.begin(9600);
 //initialize all buffers to have null values (decimal 0 = null character)
```

`

```
  for (int i=0; i<300; i++){
    buffer[i] = 0;
  }
  for(int i=0; i<max_length; i++) {
    lat[i] = 0;
    lon[i] = 0;
    time[i] = 0;
    date[i] = 0;
  }
  Serial.println("Setup Complete");
} //end "setup"


void loop(){

  //GPS PARSING CODE
  //if the implant is just starting, the time needs to be determined (via gps)
  //if theTime is 8pm, 12am, or 4am, gps data needs to be logged //CHECK WITH GROUP AND CHANGE AS NECESSARY
  if (StartUp == true || (theTime == 790 && ping_successful == false) || (theTime == 0 && ping_successful == false) || (theTime == 240 &&
ping_successful == false)){
    ping_successful = false; //show that correct gps data has not yet been successfully received
    //digitalWrite(powerGPS, ); //power up gps
    prev_millis = millis(); //used for seeing how long it takes to successfully ping the satellites

    if (StartUp == true){
      Serial.println("'StartUp' Mode: Attempting contact with GPS satellites for time-of-day initialization");
    }
    else{
      Serial.println("'Ping' Mode: Attempting contact with GPS satellites to obtain location data");
    }
    Serial.println("");
    Serial.println("Waiting for valid GPS data...");

    while (ping_successful != true){
      //check how much time has passed since beginning this loop for pinging the satellites
      curr_millis = millis();
      if (prev_millis > curr_millis){ //check for millis() overflow and adjust if necessary
        //millis() has range of 0 to 4294967295, so it overflows after reaching 4294967295
        //overall, duration in minutes needs to equal ((4294967295 + 1) - (prev_millis - curr_millis))/60000
        //because of overflow issues, the above computation needs to be carried out in the following order of steps:
        duration = 4294967295;
        duration -= (prev_millis - curr_millis);
        duration += 1;
        duration /= 60000;
      }
      else{
        duration = (curr_millis - prev_millis)/60000; //duration equals the # of min since last time calculation
      }

      //if implant is no longer in start-up mode and duration is greater than 15 min,
      //then the implant is in bad location for acquiring satellite signals, so this attempt should be terminated
      if (StartUp == false && duration > 15){
        Serial.println("");
        Serial.println("Satellite communication cannot be established. Quitting attempt.");
        break; //this will exit the while loop and skip down to the line of code that powers down the gps
      }
      //if the implant is in start-up mode, the duration to successfully acquire gps data may be > than 20 min
      //thus the while loop should continue until valid gps data has been obtained

      byteGPS=Serial.read(); //read a byte from the gps serial port
      //Serial.print(char(byteGPS));
      if (byteGPS == -1){ //check if the serial port is empty
        delay(100); //if so, wait 0.1 seconds
      }
      else{
        buffer[buffer_length] = byteGPS; //if there is serial port data, it is put in the buffer
```

`

```
buffer_length++;
if (byteGPS == 13){ //if the received byte is equal to 13, there is no more data to be received
  //buffer is now ready for parsing
  code_match = true;
  //Serial.println("");

  for (int i=1; i<7; i++){ //verify if the received command starts with $GPRMC
    if (buffer[i] != NMEA_code[i-1]){
      code_match = false; //if there is a mismatch, code_match is false
    }
  }

  indices_length = 0;
  if (code_match == true){ //if the NMEA_code is matched, continue parsing data
    //need to fill in the indices array for efficient data parsing
    for (int i=0; i<300; i++){
      if (buffer[i]==','){ //check for the position of the  "," separator
        indices[indices_length]=i;
        indices_length++;
      }
      if (buffer[i]=='*'){ //check for the position of the "*" character
        indices[12]=i;
        indices_length++;
      }
    }
    //indices array now updated
    //Serial.println("No reception yet. GPS Status is V");
    if (buffer[indices[1]+1] == 'A'){ //check if the "status" is 'A' OK. If so, then store info into buffer
      Serial.print("Total Wait Time (in Minutes): ");
      Serial.println(duration);
      Serial.println("");
      Serial.println("GPS Status is 'A': Valid GPS data obtained");
      Serial.println("");
      for (int i=0; i<12; i++){
        //CASE 0 : Time in UTC (HhMmSs)
        //CASE 1 : Status (A=OK,V=KO)
        //CASE 2 : Latitude
        //CASE 3 : Direction (N/S)
        //CASE 4 : Longitude
        //CASE 5 : Direction (E/W)
        //CASE 6 : Velocity in knots
        //CASE 7 : Heading in degrees
        //CASE 8 : Date UTC (DdMmAa)
        //CASE 9 : Magnetic degrees
        //CASE 10 : (E/W)
        //CASE 11 : Mode
        //CASE 12 : Checksum

        //store time data
        if (i == 0 && time_length < max_length){
          Serial.print("Time Data: ");
          for (int j=(indices[i]+1); j<indices[i+1]; j++){
            Serial.print(char(buffer[j]));
            time[time_length] = buffer[j];
            time_length++;
          }
          time[time_length] = end_of_instance;
          time_length++;

          //update theTime according to GPS time
          prev_millis = millis();
          int k = indices[i]+1;
          //ascii character '0' is equal to decimal value 48
          theTime = int(buffer[k+1] - 48); //buffer[k+1] is the 'h' of 'Hh'
          theTime += (int(buffer[k] - 48))*10; //buffer[k] is the 'H' of 'Hh' which explains the multiplicand of 10
          //theTime now equals the hour of the day in Coordinated Universal Time (UTC)
```

`

```
  //DANGEROUS ASSUMPTION: currently assuming that implant will remain in Central Time Zone
  theTime -= 5; //converts theTime to Central Time Zone
  if (theTime < 0){
    theTime += 24;
  }
  theTime *= 60; //converts theTime to minutes
  theTime += int(buffer[k+3] - 48); //buffer[k+3] is the 'm' of 'Mm'
  theTime += (int(buffer[k+2] - 48))*10; //buffer[k+2] is the 'M' of 'Mm'
  //theTime is now updated (assuming gps time is correct)
 }

 //store latitude data
 if (i == 2 && lat_length < max_length){
  Serial.println("");
  Serial.print("Latitude Data: ");
  for (int j=(indices[i]+1); j<indices[i+1]; j++){
    Serial.print(char(buffer[j]));
    lat[lat_length] = buffer[j];
    lat_length++;
  }
  lat[lat_length] = end_of_instance;
  lat_length++;
 }

 //store longitude data
 if (i == 4 && lon_length < max_length){
  Serial.println("");
  Serial.print("Longitude Data: ");
  for (int j=(indices[i]+1); j<indices[i+1]; j++){
    Serial.print(char(buffer[j]));
    lon[lon_length] = buffer[j];
    lon_length++;
  }
  lon[lon_length] = end_of_instance;
  lon_length++;
 }

 //store date data
 if (i == 8 && date_length < max_length){
  Serial.println("");
  Serial.print("Date Data: ");
  for (int j=(indices[i]+1); j<indices[i+1]; j++){
    Serial.print(char(buffer[j]));
    date[date_length] = buffer[j];
    date_length++;
  }
  date[date_length] = end_of_instance;
  date_length++;
 }
} //end "for (int i=0;i<12;i++)"

Serial.println("");
Serial.print("GPS Time: ");
Serial.print(theTime);
Serial.print(" min. = ");
int tt = theTime/60;
int mm = theTime - (tt*60);
Serial.print(tt);
Serial.print(":");
if (mm <= 9){
 Serial.print(0);
}
Serial.print(mm);
Serial.println(" CST");

ping_successful = true; //confirm that gps data was successfully received
```

`

```
            if (StartUp == true){
              StartUp = false; //start-up time has been determined, implant is no longer in start-up mode
            }

          } //end "if (buffer[indices[1]+1] == 'A')"
        } //end "if (code_match == true)"

        //reset buffer
        buffer_length = 0;
        for (int i=0; i<300; i++){
          buffer[i] = 0;
        }

      } //end "if (byteGPS == 13)"
    } //end "if (byteGPS == -1) else"
  } //end "while (ping_successful != true)"

  //digitalWrite(powerGPS, ); //power down gps
  Serial.println("");
  Serial.println("Waiting for next GPS ping or Base Station data transfer...");
} //end "if (StartUp == true || theTime == 1200 || theTime == 0 || theTime == 240)"


//TIME MONITORING CODE
curr_millis = millis();
if (prev_millis > curr_millis){ //check for millis() overflow and adjust if necessary
  //millis() has range of 0 to 4294967295, so it overflows after reaching 4294967295
  //overall, duration in minutes needs to equal ((4294967295 + 1) - (prev_millis - curr_millis))/60000
  //because of overflow issues, the above computation needs to be carried out in the following order of steps:
  duration = 4294967295;
  duration -= (prev_millis - curr_millis);
  duration += 1;
  duration /= 60000;
}
else{
  duration = (curr_millis - prev_millis)/60000; //duration equals the # of min since last time calculation
}
if (duration > 0){
  theTime += duration;
  theTime = theTime % 1440; //cycle through the 1440 minutes in 1 day
  //theTime is now updated
  //if theTime == 0, this means the time is 12am
  prev_millis = curr_millis;
  //Serial.println(duration);
  //Serial.println(theTime);
  ping_successful = false;
}


//BASE STATION COMMUNICATION CODE
if (transmit_successful == true){
  if (transmit_time <= theTime){
    if ((transmit_time + 60) == theTime){
      transmit_successful = false;
    }
  }
  if (transmit_time > theTime){
    if ((transmit_time + 60) == (theTime + 1440)){
      transmit_successful = false;
    }
  }
}

byteIn = transSerial_R.read(); //read a byte from the transceiver
if (byteIn == base_signal && transmit_successful == false){ //check if the base station has been found. if so, send response and all data
  Serial.println("Base Station found. Preparing for data transfer");
```

`

```
    digitalWrite(TR_Pin, HIGH); //set transceiver to transmit
    pinMode(12, OUTPUT); //set transSerial_T tx pin as digital output

    //send response 6 times so the base station will not accidentally miss it
    for(int i=0; i<6; i++){
      transSerial_T.write(implant_response);
    }

    Serial.println("Sending data...");
    //send all stored data (in the message format) 3 times for ensured data transmission
    transSerial_T.write(byteStart);
    for( int i=0; i<3; i++){
      for(int i=0; i<lat_length; i++){
        transSerial_T.write(lat[i]);
      }
      transSerial_T.write(byteEnd);
      for(int i=0; i<lon_length; i++){
        transSerial_T.write(lon[i]);
      }
      transSerial_T.write(byteEnd);
      for(int i=0; i<time_length; i++){
        transSerial_T.write(time[i]);
      }
      transSerial_T.write(byteEnd);
      for(int i=0; i<date_length; i++){
        transSerial_T.write(date[i]);
      }
      transSerial_T.write(byteEnd);
      transSerial_T.write(byteStop);
    } //end "for( int i=0; i<3; i++)"
    transSerial_T.write(byteStart);
    Serial.println("Data transfer complete");
    //reset lat, lon, time, and date buffers
    lat_length = 0;
    lon_length = 0;
    time_length = 0;
    date_length = 0;
    for (int i=0; i<max_length; i++){
      lat[i] = 0;
      lon[i] = 0;
      time[i] = 0;
      date[i] = 0;
    }
    digitalWrite(TR_Pin, LOW); //set transceiver back to receive
    pinMode(12, INPUT); //set transSerial_R rx pin as digital input
    transmit_successful = true;
    transmit_time = theTime;

    Serial.println("");
    Serial.println("Waiting for next GPS ping or Base Station data transfer...");
  } //end "if (byteIn == base_signal)

} //end "loop"
```

`

Appendix E Base Station Arduino Code

```
#include <SoftwareSerial.h>
#include <string.h>
#include <ctype.h>  //are these first 2 lines necessary? (I saw them in sample code online for GPS parsing)

int rxPin = 0; //serial input pin
int txPin = 1; //serial output pin
int trPin = 2; //digital output pin that connects to transceiver's T/R SEL pin
SoftwareSerial dlSerial(3, 4); //3 is rx, 4 is tx
int count = 0; //used for loading indexing buffer
char byteIn = 0; //variable for temporarily storing incoming byte
char buff[300]; //input buffer for storing received bytes from implant (the size is arbitrarily 300, can be changed)

const int max_length = 100;
char lat[max_length]; //buffer for storing latitude values
char lon[max_length]; //buffer for storing longitude values
char time[max_length]; //buffer for storing time-stamp values
char date[max_length]; //buffer for storing date-stamp values
int lat_length = 0; //gives the length of lat being used
int lon_length = 0; //gives the length of lon being used
int time_length = 0; //gives the length of time being used
int date_length = 0; //gives the length of date being used

char byteStart = 1; //the start byte is a "start of heading" character
char byteEnd = 10; //the end byte is a "new line" character
char byteStop = 4; //the stop byte is a "end of transmission" character
char end_of_instance = 13; //used to separate instances of gps data (decimal 13 = 'carriage return')
//MESSAGE FORMAT: byteStart, (latitudes, byteEnd, longitudes, byteEnd, times, byteEnd, dates, byteEnd, byteStop)x3, byteStart

boolean tansfer_successful = false;
unsigned long curr_millis = 0;
unsigned long prev_millis = 0;
unsigned long duration = 0;

void setup() {
  pinMode(rxPin, INPUT);
  pinMode(txPin, OUTPUT);
  pinMode(trPin, OUTPUT);
  Serial.begin(9600); //set baud rate to 9600 (transceiver data sheet says it can handle up to 10,000 baud rate)
  for(int i=0; i<300; i++) {
    buff[i] = 0; //initialize all buffer characters to NULL value
  } //end "for"
  for(int i=0; i<max_length; i++) {
    lat[i] = 0;
    lon[i] = 0;
    time[i] = 0;
    date[i] = 0;
  }
} //end "setup"

void loop() {
```

`

```
digitalWrite(trPin, HIGH); //set transceiver to transmit mode
for(int i=0; i<5; i++) {
  Serial.write(63); //send detection signal 5 times
} // end “for”


if (transfer_successful == true){
  curr_millis = millis();
  if (prev_millis > curr_millis){ //check for millis() overflow and adjust if necessary
    //millis() has range of 0 to 4294967295, so it overflows after reaching 4294967295
    //overall, duration in minutes needs to equal ((4294967295 + 1) - (prev_millis - curr_millis))/60000
    //because of overflow issues, the above computation needs to be carried out in the following order of steps:
    duration = 4294967295;
    duration -= (prev_millis - curr_millis);
    duration += 1;
    duration /= 60000;
  }
  else{
    duration = (curr_millis - prev_millis)/60000; //duration equals the # of min since last time calculation
  }
  if(duration > 60){
    transfer_successful = false;
  }
}

digitalWrite(trPin, LOW); //set transceiver to receive mode
for(int i=0; i<7; i++) {
  if(transfer_successful == false){
    byteIn = Serial.read();
    Serial.println(byteIn);
    if(byteIn == 33) {  //check if the received byte equals the response signal

      //look for the start byte (some of the bytes being read may still be the response signal, 45)
      while(byteIn != byteStart || byteIn != byteStop) {
        byteIn = Serial.read();
        Serial.print(byteIn);
      } //end “while"
      Serial.println("");

      byteIn = Serial.read();
      //look for the end byte and store all previous bytes in the latitude buffer, lat[]
      while(byteIn != byteEnd && lat_length < 150) {
        Serial.print(char(byteIn));
        lat[lat_length] = byteIn;
        lat_length++;
        byteIn = Serial.read();
      } //end “while”
      Serial.println("");

      byteIn = Serial.read();
      //look for the end byte and store all previous bytes in the longitude buffer, lon[]
      while(byteIn != byteEnd && lon_length < 150) {
        Serial.print(char(byteIn));
        lon[lon_length] = byteIn;
        lon_length++;
        byteIn = Serial.read();
      } //end “while”
      Serial.println("");

      byteIn = Serial.read();
      //look for the end byte and store all previous bytes in the time buffer, time[]
      while(byteIn != byteEnd && time_length < 150) {
        Serial.print(char(byteIn));
        time[time_length] = byteIn;
        time_length++;
        byteIn = Serial.read();
```

`

```
} //end “while”
Serial.println("");

byteIn = Serial.read();
//look for the end byte and store all previous bytes in the date buffer, date[]
while(byteIn != byteEnd && date_length < 150) {
  Serial.print(char(byteIn));
  date[date_length] = byteIn;
  date_length++;
  byteIn = Serial.read();
} //end “while”
Serial.println("");

transfer_successful = true;
prev_millis = millis();

Serial.println("Writing data to USB stick");

//open file
dlSerial.write("OPW ");
dlSerial.write("GPSdata.txt");
dlSerial.write(13);

//write time data to file
dlSerial.write("WRF ");
dlSerial.write(6);
dlSerial.write(13);
dlSerial.write("Time: ");
dlSerial.write(13);
delay(1000);
for (int i = 0; i < time_length; i++){
  dlSerial.write("WRF ");
  dlSerial.write(1);
  dlSerial.write(13);
  dlSerial.write(char(time[i]));
  dlSerial.write(13);
  delay(1000);
}

//write date data to file
dlSerial.write("WRF ");
dlSerial.write(6);
dlSerial.write(13);
dlSerial.write("Date: ");
dlSerial.write(13);
delay(1000);
for (int i = 2; i < 4; i++){
  dlSerial.write("WRF ");
  dlSerial.write(1);
  dlSerial.write(13);
  dlSerial.write(char(date[i]));
  dlSerial.write(13);
  delay(1000);
}
dlSerial.write("WRF ");
dlSerial.write(1);
dlSerial.write(13);
dlSerial.write("/");
dlSerial.write(13);
delay(1000);
for (int i = 0; i < 2; i++){
  dlSerial.write("WRF ");
  dlSerial.write(1);
  dlSerial.write(13);
  dlSerial.write(char(date[i]));
  dlSerial.write(13);
```

`

```
  delay(1000);
}
dlSerial.write("WRF ");
dlSerial.write(1);
dlSerial.write(13);
dlSerial.write("/");
dlSerial.write(13);
delay(1000);
for (int i = 4; i < (date_length-1); i++){
  dlSerial.write("WRF ");
  dlSerial.write(1);
  dlSerial.write(13);
  dlSerial.write(char(date[i]));
  dlSerial.write(13);
  delay(1000);
}
dlSerial.write("WRF ");
dlSerial.write(1);
dlSerial.write(13);
dlSerial.write(10);
dlSerial.write(13);
delay(1000);

//write lat and lon data in hyperlink
dlSerial.write("WRF ");
dlSerial.write(11);
dlSerial.write(13);
dlSerial.write("http://www.");
dlSerial.write(13);
delay(1000);
dlSerial.write("WRF ");
dlSerial.write(11);
dlSerial.write(13);
dlSerial.write("maps.google");
dlSerial.write(13);
delay(1000);
dlSerial.write("WRF ");
dlSerial.write(12);
dlSerial.write(13);
dlSerial.write(".com/maps?q=");
dlSerial.write(13);
delay(1000);
for(int i=0; i<(lat_length-1); i++){
  dlSerial.write("WRF ");
  dlSerial.write(1);
  dlSerial.write(13);
  dlSerial.write(char(lat[i]));
  dlSerial.write(13);
  delay(1000);
}
dlSerial.write("WRF ");
dlSerial.write(1);
dlSerial.write(13);
dlSerial.write(",");
dlSerial.write(13);
delay(1000);
for(int i=0; i<(lon_length-1); i++){
  dlSerial.write("WRF ");
  dlSerial.write(1);
  dlSerial.write(13);
  dlSerial.write(char(lon[i]));
  dlSerial.write(13);
  delay(1000);
}
dlSerial.write("WRF ");
dlSerial.write(1);
```

`

```
        dlSerial.write(13);
        dlSerial.write(10);
        dlSerial.write(13);
        delay(1000);

        //close file
        dlSerial.write("CLF ");
        dlSerial.write("GPSdata.txt");
        dlSerial.write(13);

        Serial.println("Done writing to USB stick");

        lat_length = 0;
        lon_length = 0;
        time_length = 0;
        date_length = 0;
        for (int i=0; i<max_length; i++){
          lat[i] = 0;
          lon[i] = 0;
          time[i] = 0;
          date[i] = 0;
        }

      } //end “if”
    } //end "if"
  } //end “for”

} //end “loop”
```