IROTS

By

Bilal Gabula

Gerard McCann

Osayanmo Osarenkhoe

Final Report for ECE 445, Senior Design, Fall 2012

TA: Justine Fortier

12 December 2012

Project No. 15

Abstract

The paper describes the design and verification of a river otter tracking system. The system consists of an implant and a base station. The implant periodically stores its latitude, longitude, and time of acquisition of aforementioned position coordinates. The implant will transfer the data to the base station when within range. The base station will serve as the location that the researcher can access the data and easily transfer the data to their computer. The latitude, longitude and acquisition time will be determined using a standard Global Positioning System while the wireless data transfer is over a generic sub-1 gigahertz Radio Frequency system. The original proposal included a piezoelectric power generation scheme that is proven to be impossible with the technology used.

Contents

1. INTRODUCTION

The Illinois River otter tracking system is a project primarily dedicated to the improvement of researcher's ability to perform studies that involve animal tracking. The Illinois River otters are amazing creatures, extremely active, curious and playful. Working on a device that will improve our understanding of such enjoyable animals has been very fulfilling on many fronts. This project was necessary due to the lack of preexisting tracking devices suitable for otters. This is primarily due to their "strange" anatomy. Where the otters neck is broader than its head which makes using a GPS collar impractical. Our main objective was to create a host (Otter, or other animal to be tracked) friendly device by making the device as small as possible to reduce irritation due to the implantation. In order to achieve this, we had to design the device starting from the printed circuit board (PCB), and then including individual integrated circuits (ICs). This also introduced the need for low power components so as to minimize the size of our power electronics and battery. We also aimed to make the device user (researcher) friendly by eliminating the need to recapture the otters for data extraction. To this end, we designed the device to automatically transfer data to a base station located in an area the otters are known to visit frequently. We were able to attain a variable mapping definition by giving the researcher control over the data acquisition frequency. The project posed several additional degrees of complexity due to product life, and data accuracy requirements.

In the following chapters, we will be describing the methods and thought processes that went into the designing of our final device. We will give an in depth analysis of the tests we performed to verify our design choices, as well as an explanation of the results of these tests. Also, we will take a look at the outcome of our efforts so far, as well as future modifications which could bring us closer to our design goal of an implantable GPS tracking device.

2 DESIGN

2.1 BLOCK DIAGRAM

Note: µC ⇒ Microcontroller; Note: RF ⇒ Radio Frequency

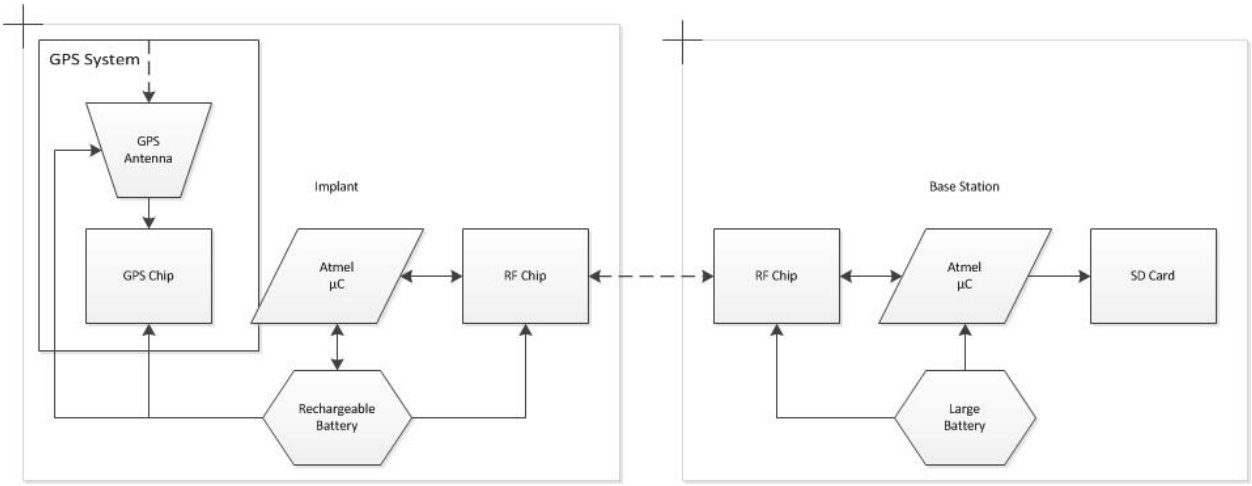


Figure 1: Block diagram of complete system

2.2 Block Description

2.2.1 Implant

GPS System**:** This system acquires the GPS location data from the satellites. This data includes the time stamp, Longitude, and Latitude. The altitude is acquired if possible, but not necessarily. It receives power from the rechargeable battery.

Implant µC**:** This is the processing center of the implants. It controls power consumption and data storage. It receives the data from the GPS system and stores it. It receives battery level information from the Rechargeable Battery using and uses this information, along with its internal clock, to control the RF chip and GPS system.

RF Chip & Antenna: This relays the stored data from the µC to the base station. It receives its power from the battery in the Rechargeable Battery.

Rechargeable Battery: This provides power to the complete implant system including the GPS system, Implant Microcontroller and the RF chip.

2.2.2 Base Station

Large Battery**:** This battery will supply power to the other components of the base station. It will be rechargeable and easily changeable.

RF Chip: This receives the relayed data from the implant and sends it to the µC on the base station. It receives its power from the battery in the large battery.

Base Station µC**:** This is the processing center of the base station. It controls power consumption and data storage. It receives the data from the RF chip and stores it. It receives power from the large battery.

SD Connector**:** This is used to transfer the stored data to the SD card when data is being retrieved by the researcher. It gets its power from the large battery, and its data from the µC.

2.3 Schematics

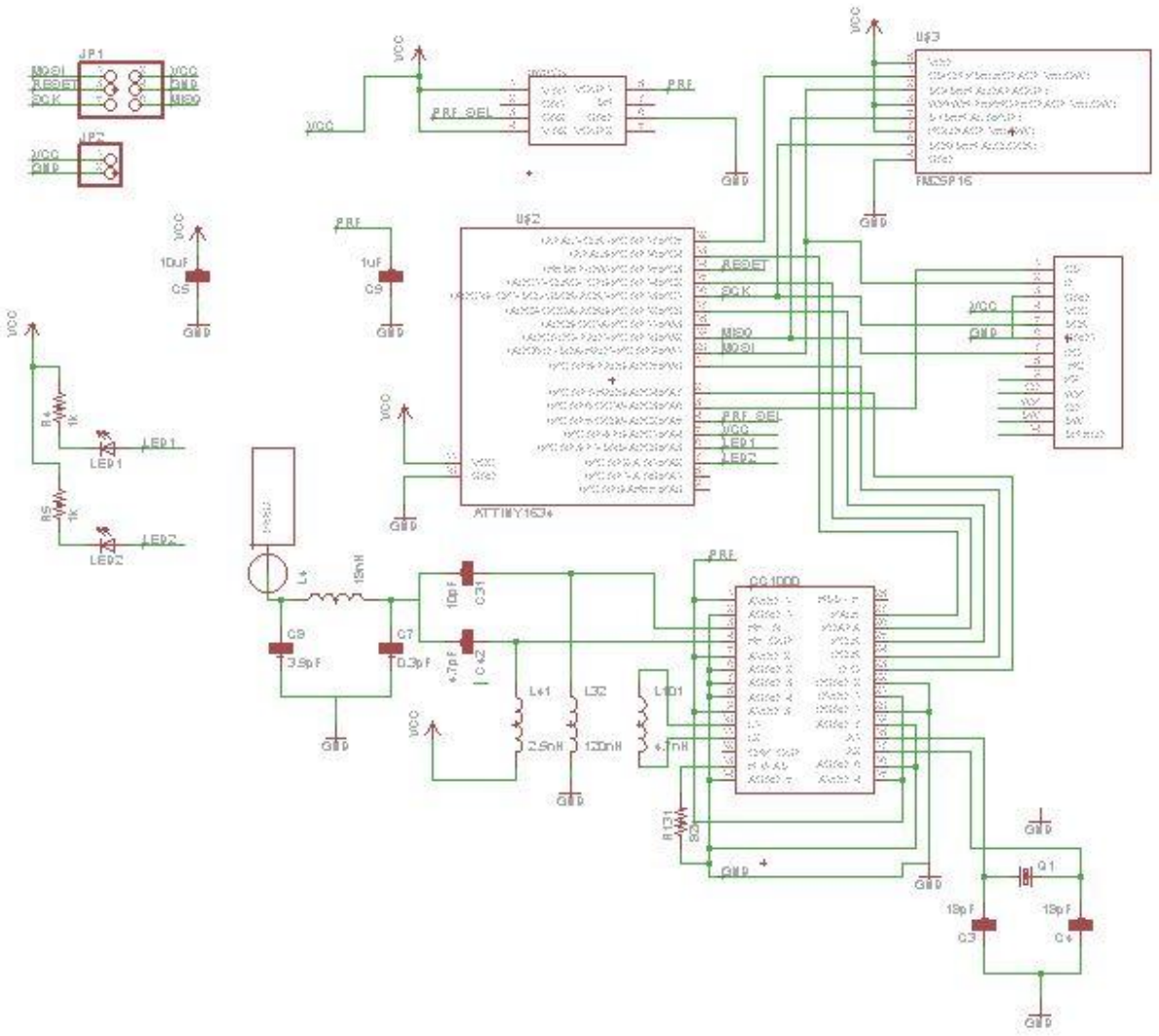


Figure 2: Base Station Schematic

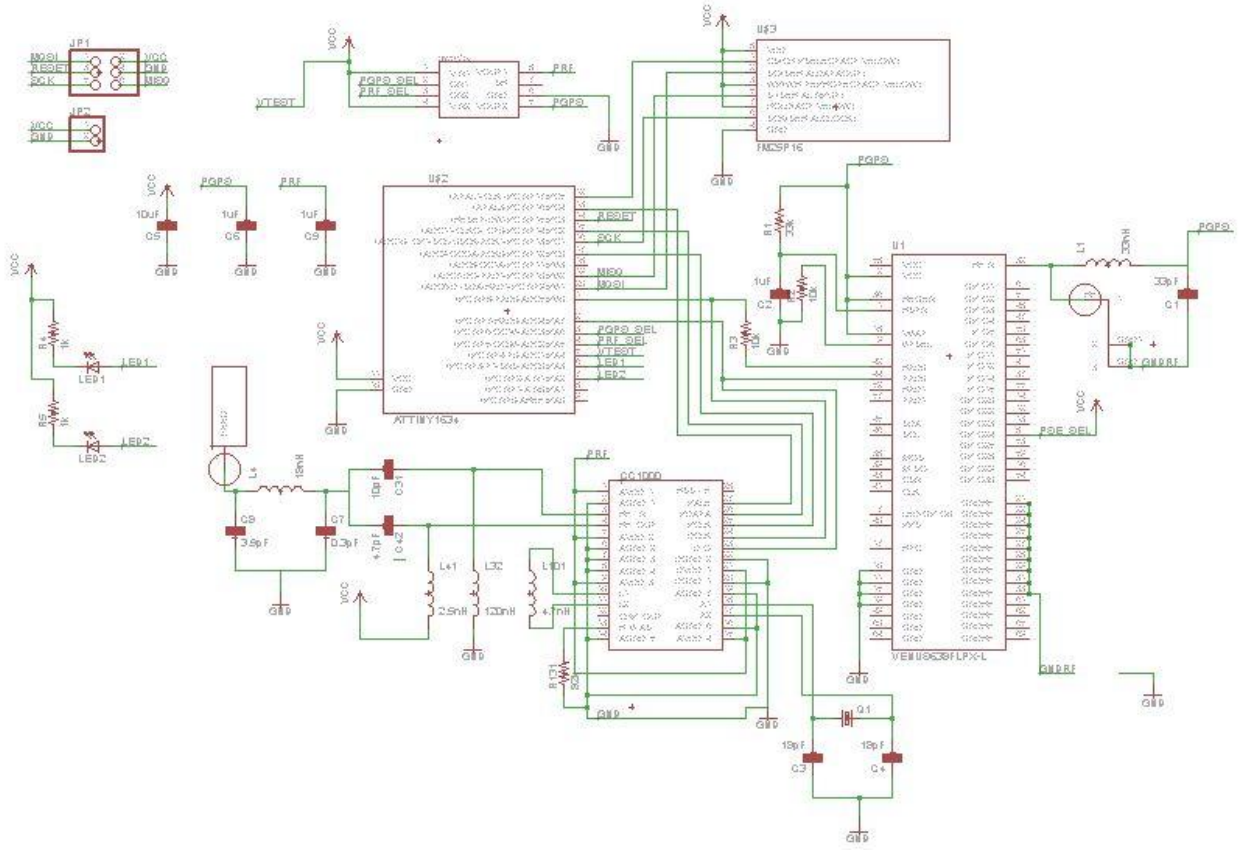


Figure 3: Implant Schematic

2.4 Schematic Description

Implant

GPS System: [1] This system consists of the GPS antenna (Sarantel SL1204) and the GPS chip (Skytrak Venus638FLPx-L). The Antenna signal goes through a small filter network [1] and goes into the RF_{in} of the GPS chip. The other two pins on the antenna are connected to ground. The GPS chip has communication pins RXD0 and TXD0 that are connected to the TX and RX pins of the microcontroller respectively. Power has to be supplied to the GPS chip to the VCC and VBAT pins which is done using the signal called GPS_BAT so that the microcontroller can completely shut off the GPS chip to conserve power.

Implant µC: [2] Other than the connections mentioned above the microcontroller (Atmel ATtiny 1634) is connected to the RF chip and the Rechargeable Battery and Charging System. The RF chip connections are discussed in the RF chip and Antenna section of the Schematic description. The signals BAT_RF_EN, BAT_GPS_EN are signals that enable power to the RF and GPS chips and antennas respectively. BAT_MIC is connected to VCC of the implant microcontroller and is the supply voltage out of the rechargeable Battery and Charging system. The BAT_RD_EN signal enables the BAT_RD signal to have a voltage that is a fraction of the battery voltage so that the µC can read the battery voltage and estimate the amount of power left.

Rechargeable Battery: [3-4] The Unionfortune PRT-00339 is a 1000mAh rechargeable polymer lithium ion battery.

RF Chip & Antenna: [5] The antenna filter network for the RF antenna (Johanson 868 MHz Antenna) is based off the design from the datasheet [5]. The external oscillator is to maintain the internal clocks so as to encode and decode RF signals. At the RF bias pin a high-precision resistor is connected for the band gap of the internal system. The inductor L1 is the high-precision inductor for the internal tank circuit. The connections PALE, PDATA and PCLK are communication pins to set up the operating modes of the chip (TI CC1000). They are connected to the microcontroller GPIO pins as the chip does not follow any standard communication protocol. Similarly the DCLK and DIO are the communication pins for data transfer and are connected to GPIO pins of the µC.

Base Station

Large Battery: This will be an Energizer 9V battery. The positive terminal will go into the 7805 to be regulated and the negative terminal will be considered as ground for the internal circuit.

RF Chip: The TI CC1000 will be hooked up almost exactly the same as in the implant.

Base Station µC: The connections from the microcontroller (Attiny 1634) to the RF chip are exactly the same. VCC is connected to the output of the line regulator 7805. The SD card connector is connected to the µC via GPIO pins so as to implement the FAT32/16 library to write files onto the SD card connected.

SD Card: This is a female SD connector to enable easy connection of a portable SD card to the solder pads of the Base Station µC

2.5 Code Block Diagram

The code block diagrams are shown below. All of the code will be written in C and commented thoroughly. The compiler will be avr-gcc (from win-avr). We will be using an in-system-programmer (ISP) called USB-ASP. Since we were unable to completely test all of the components of the implant and base station we haven't compiled code that would implement this flow chart.

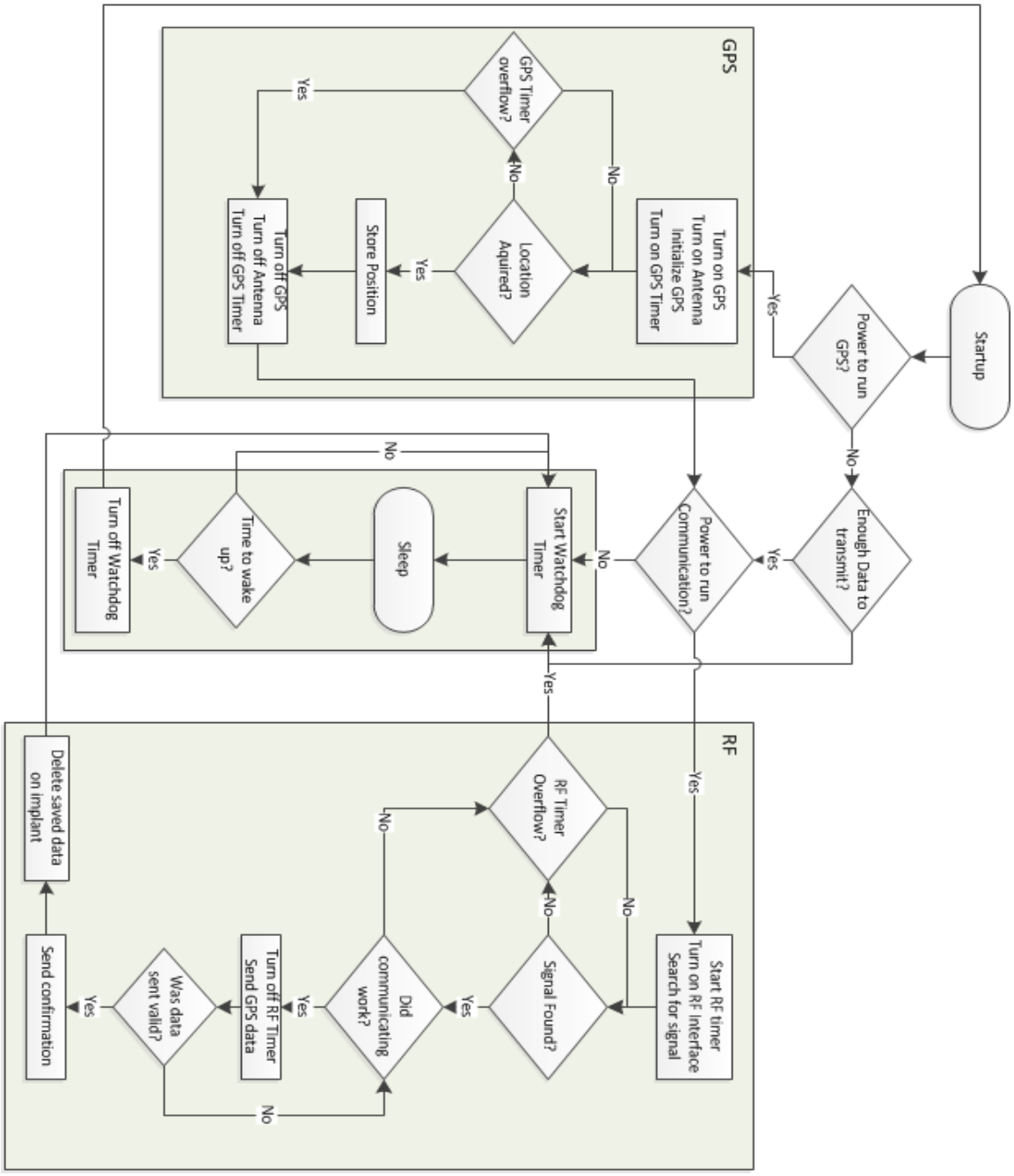


Figure 4: Code Flowchart 1 – Implant

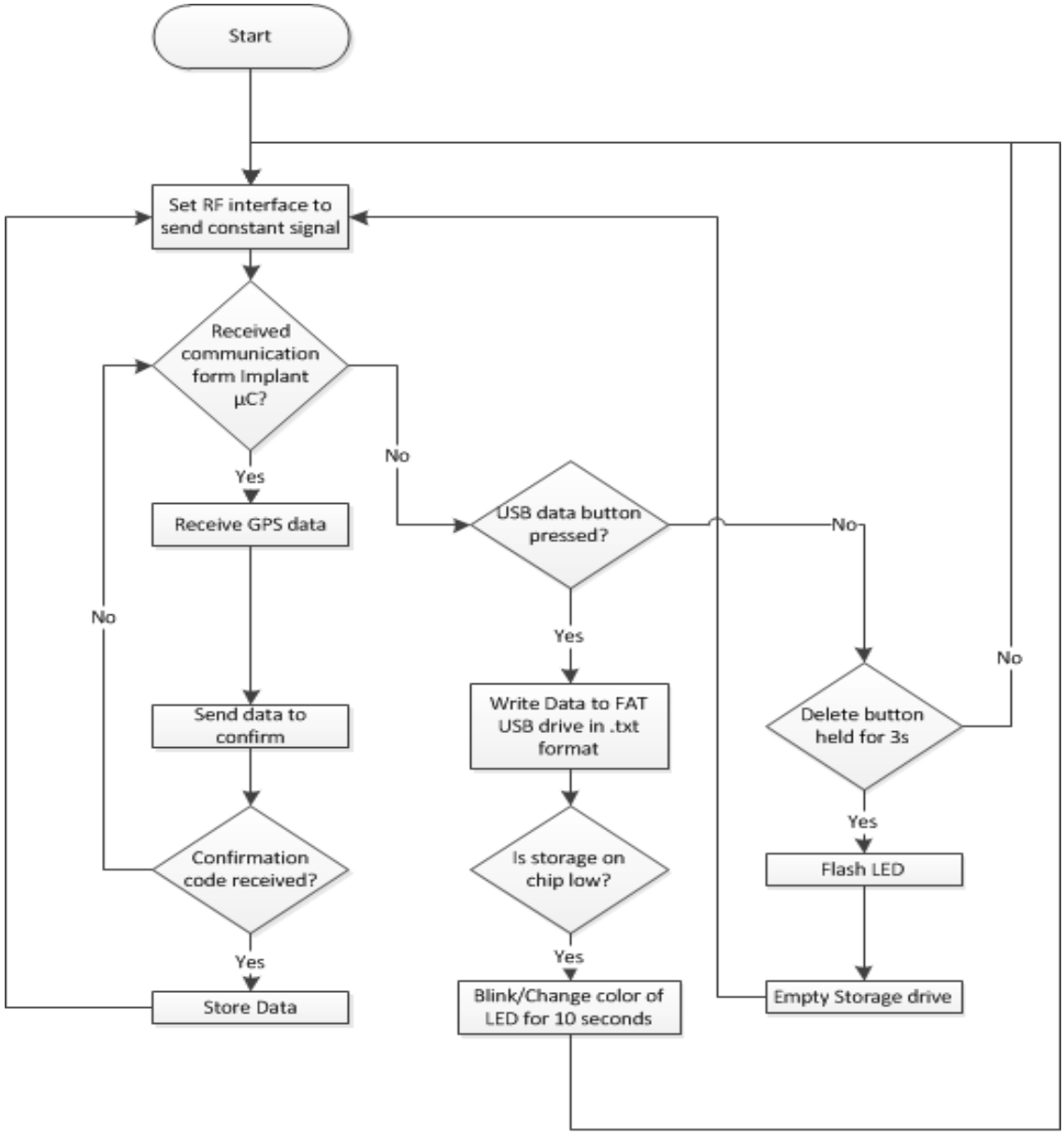


Figure 5: Code Flowchart 2 – Base Station

3. DESIGN VERIFICATION

The first step in the testing process was to test each chip individually on a breakout board made from the parts shop. The description of the breakout boards constructed and the test results are shown in Appendix C.

3.1 GPS Test Results

Using the test breakout board we were able to test a sample of the GPS chip which we will use. The antenna used was a patch antenna and was not under open sky thus giving us a less accurate, less time efficient location fix (compared to our results when using the proposed GeoHelix antenna). We started by testing communications with the chip. Using an LCD screen we were able to display the ASCII (NMEA) and the Hex (Skytraq Binary™) output. Since the NMEA output has a higher space requirement we decided to use the binary output as conversion would just add time/power requirements. Here are some readings we were able to obtain from the GPS chip.

Name of Value	Format	Scale	Hex Value read	Scaled value
Latitude*	SINT32	10^{-7} degrees	0x17e8d479 0x17e8d470	40.1134713°N 40.1134704°N
Longitude*	SINT32	10^{-7} degrees	0xcb6a26e4 0xcb6a20ee	88.2235676°W 88.2237202°W
Week number	UINT16	Counted up from 1/6/1980	0x06ac	1708 ⇨Week of 9/30/2012
Time of week	UINT32	10^{-2} seconds	0x13776b8	2041220.88 seconds ~9am Tuesday GMT

Table 1: Readings from GPS chip

*From two different cold starts and acquisition within 5 minutes.

Figure 6: Google Maps image of the locations acquired (two blue pins) and the actual position (approximately green pin).

The distance between the measurements and the actual location is very small compared to the required accuracy.

3.2 Power Generation Test Results

To test the power generation circuit, we connected the piezoelectric device to our rectification circuit (a more detailed description can be found in the design section). Using light taps that we estimated would simulate the minimum vibrational energy due to each step by the otter, we hit the piezoelectric device several times to get a good average of the energy we could harvest. The voltage across the capacitor was measured before and after each hit, and the voltage difference was used to calculate the energy stored. Below are tables with our results from the piezoelectric crystal, and from the piezoelectric energy generator.

Initial Voltage (V)	Voltage after hit (V)	Energy from hit (J)
4	4.11	4.46E-07
0.1	0.134	3.98E-09
0.09	0.23	2.24E-08
0.26	0.378	3.76E-08
0.35	0.549	8.95E-08
0.52	0.924	2.92E-07
0.92	0.97	4.73E-08
0.0109	0.29	4.20E-08
0.072	0.182	1.40E-08

0.15	0.18	4.95E-09
0.17	0.2	5.55E-09
0.185	0.52	1.18E-07
0.035	0.048	5.40E-10
0.042	0.075	1.93E-09
0.065	0.084	1.42E-09
0.075	0.085	8.00E-10
0.132	0.154	3.15E-09
0.149	0.179	4.92E-09
0.174	0.204	5.67E-09

Table 2: Piezoelectric crystal energy data

The average power generated from a hit is $6.03*10^{-8}$J. This implies that if we need ~550J of energy a day, the piezoelectric crystal will need to vibrate $9.04*10^{9}$ times. This result is unusable.

Initial Voltage (V)	Voltage after hit (V)	Energy from hit (J)
1.2	1.22	1.14E-06
1.05	1.087	1.86E-06
1.5	1.57	5.05E-06
1.6	1.67	5.38E-06
3.52	3.535	2.49E-06
3.44	3.46	3.24E-06
3.455	3.473	2.93E-06
3.42	3.45	4.84E-06
3.92	3.93	1.84E-06

Table 3: Piezoelectric energy generator data

The average power generated from a hit is $3.20*10^{-6}$J. This implies that if we need ~550J of energy a day, the piezoelectric crystal will need to vibrate $1.70*10^{8}$ times. The results of this test show that the energy generator, although 90 times more efficient than the crystal still would give us unusable results.

3.3 Radio Frequency Transmitter Test

The RF communication between the base station and the implant did not work. We were not able to implement a full system test as described below, so the main RF test (Appendix A Test III.1) communication between the base station and the implant could not be performed. Partial functionality tests were performed as well to determine the how close the RF system was to working. The RF power pins (1,5,19,21 CC1000) were getting 3.33 volts when the power was connected and the RF chip GND pins (2, 6, 7, 8, 14, 15, 16, 19, 22) were getting 0 V relative to system GND. This satisfied RF Test III.1A. The RF transmission Test III.1B involved looking at the RF out (pin 4 CC1000) to see if any data was being modulated to the correct RF frequency. We used a vector signal analyzer to measure the output and the result is included below.

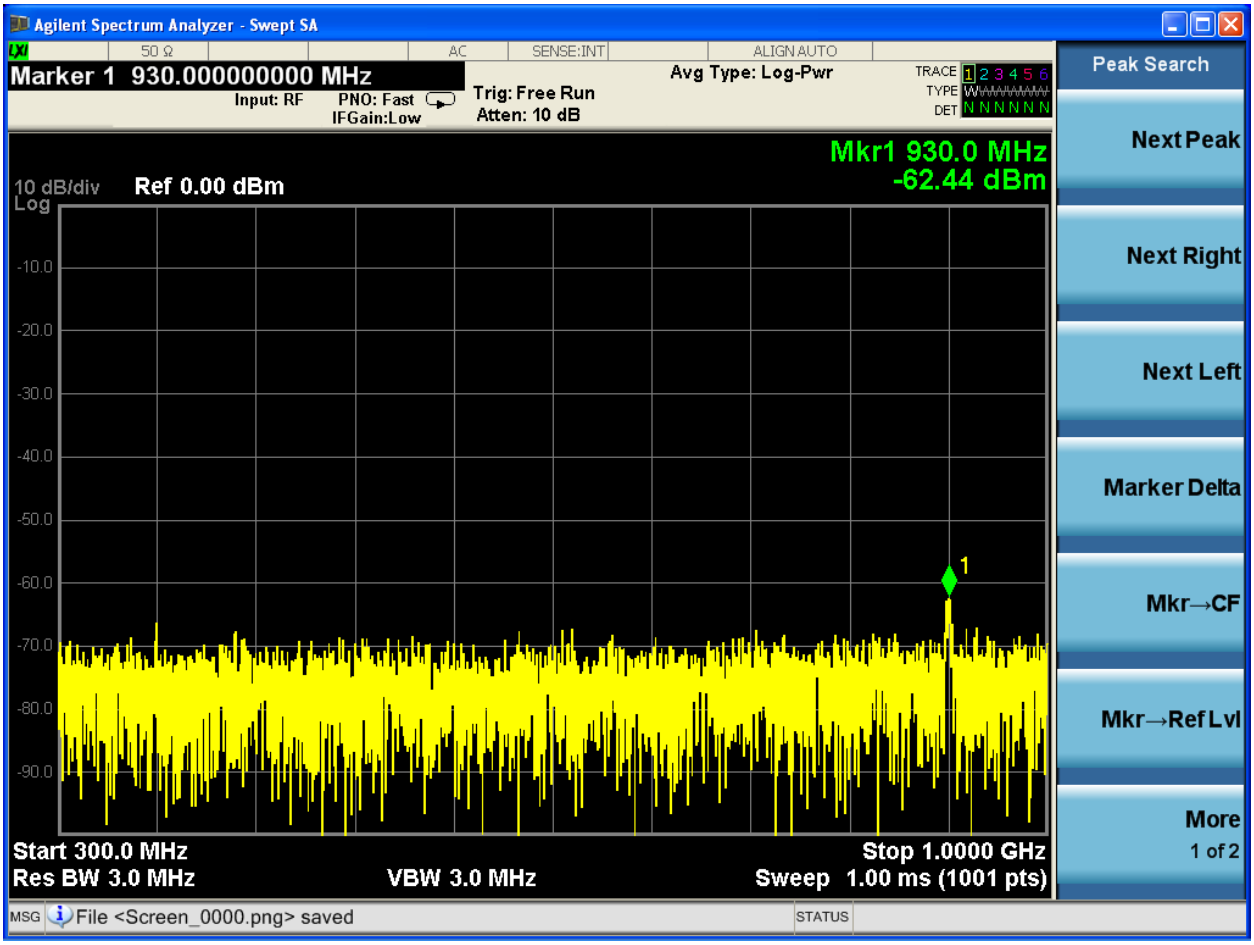


Figure 7: Output of the RF chip in transmit mode

There is a peak at 930 MHz this was promising but it turned out to be a random signal as after power to the RF chip was disconnected the peak still persisted.

Following this a test was performed to confirm the registers were being correctly initialized. Appendix Z contains the data that was sent to each register (this data was obtained from a Texas

instruments application note). The data was then read from each register and compared against the values in the appendix. They were the same. The next Tests III.1.C and III.1D failed as they require that Test III.1 B works. The next Tests III.2 and III.3 are the power requirements in transmit and receive mode. In receive mode the power consumption was 9.6 mA while in transmit mode the consumption was 16.5 mA. These values are within the specifications of 10 mA and 17 mA. They are also expected as the data sheets specifies the expected current consumption values for different power settings.

The likely problems that caused the cc1000 not to output a correct frequency were likely incorrect values in the configuration registers. With more time this would be fixed. Other possible problems could be the matching circuit. If the matching circuit components were not correct they would have prevented the chip from outputting a correct signal. Also as described in the full system test below the Rx and Tx lines to the cc1000 and the µC were on the same line as the Rx and Tx lines to the GPS chip. The GPS chip's Tx and Rx could have been drawing the lines low during communication.

3.4 Microcontroller Test

Referring to verification Test II.2 from Appendix A. We were able to communicate with using an external USB to serial port convertor (SILABS - CP2101) and Matlab we were able to verify the communication at 4800 BAUD. Here is a screenshot of the data displayed in Matlab after receiving the GPS time data from the microcontroller using UART at 4800 BAUD.

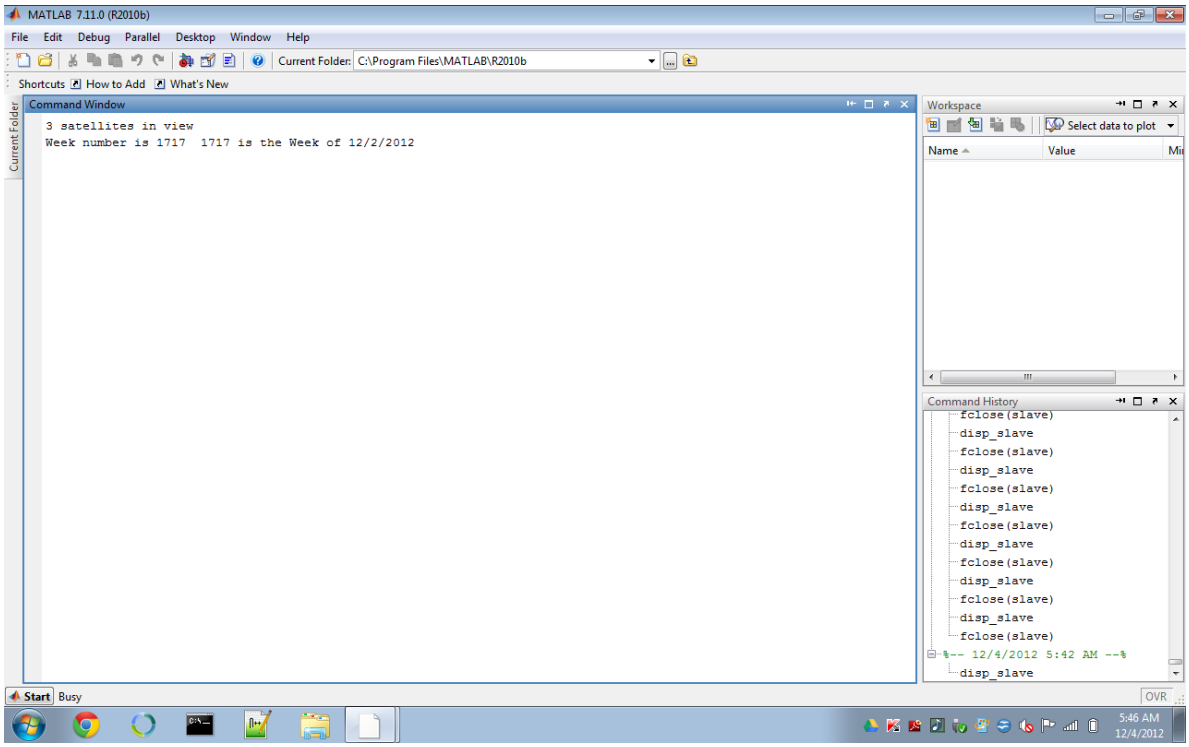


Figure 8: Microcontroller test screen shot; MATLAB output

3.5 Flash Memory Test

Referring to verification Test II.1 from Appendix A. Instead of using a combination of volatile and non-volatile memory present on the microcontroller we decided to have a dedicated memory chip (FM25P16_ds). There are two parts to this test. First, write a random value (here 0xAB) to a given register (here 0x0003). Second, turn of the power to the flash chip to restart it and read the value in the register (here 0x0003). The test was considered successful as the value read from the register was the value written to the register.

3.6 Rechargeable Battery Test

Referring to verification tests IV.1 and IV.2 from Appendix A. We ran two different tests on the battery under the same setup. The difference is in the values of R1 (30Ω and 12Ω) as shown in the Rechargeable Battery Test schematic shown below.

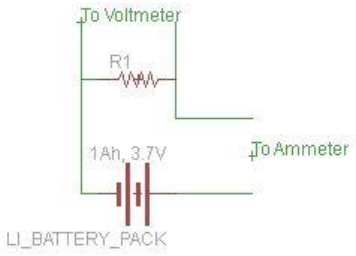


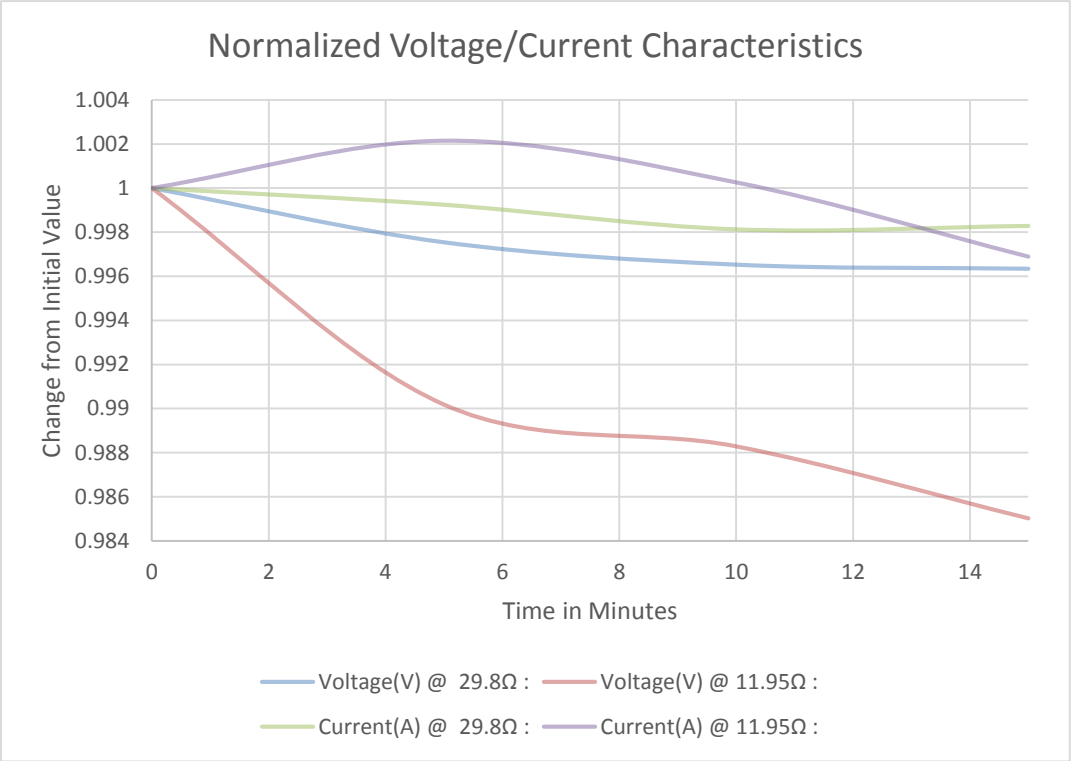
Figure 9: rechargeable battery test schematic

R1 = 29.8Ω	0 Minutes	5 Minutes	10 Minutes	15 Minutes
Voltage(V) @ 29.8Ω :	3.741	3.73182	3.72803	3.72734
Current(A) @ 29.8Ω :	0.127412	0.127316	0.127174	0.127194

Table 4: battery charging test cycle 1

1 = 11.95Ω	0 Minutes	5 Minutes	10 Minutes	15 Minutes
Voltage(V) @ 11.95Ω :	3.632	3.59634	3.58946	3.5776
Current(A) @ 11.95Ω :	0.312	0.31267	0.31208	0.31103

Table 5: battery charging test cycle 2



Graph 1: Normalized voltage/current characteristics

3.7 Power Use Test

On the final Implant board we conducted a Power Use test. The code is shown in Power Test in Appendix B and this verifies Tests I.3 and II.3 shown in Appendix A. The results were as follows.

@3.3V
GPS on, GPS antenna on, Microcontroller on: 113mA
GPS off, GPS antenna off, Microcontroller on: 20mA
GPS off, GPS antenna off, Microcontroller in sleep mode: 19mA
Therefore the GPS chip uses ~80mA. (13mA is used by the GPS antenna)
And the microcontroller uses less power during sleep mode than during active mode.

3.8 Full System Test

Since we did not have all the components of the implant and base station tested we could not perform a comprehensive test. The test we performed as a full system test was on the final implant board shown in Appendix G. We were able to get a GPS location which was accurate to within a 100 meters in 5 minutes. We displayed this result in Matlab to confirm.

4. COST

Project Total: $43527.81

4.1 Costs Analysis

Parts	Unit price ($)	Quantity (#)	Total cost ($)
Attiny 1634 (Microcontroller)[1]	1.80	2	3.60
Ramtron FM25P16 (Flash Chip)	4.29	1	4.29
Venus638FLPx-L (GPS chip)[1]	39.95	1	39.95
GeoHelix GPS Antenna[7]	22.95	1	22.95
SD Card Female connector	3.30	1	3.30
TI CC1000[5] (RF transceiver)	7.17	2	7.17
TI TPS22960(Power Switch)	1.17	1	1.17
Johanson 868 MHz antenna(RF Antenna)	1.50	2	3.00
Polymer Lithium Ion Battery - 1000mAh	6.95	1	6.95
Energizer 9V battery	4.95	1	4.95
1 GB SD Card	5.99	1	5.99
PCB main board (including shipping)	229.00	1	150
TOTAL:			327.81

Table 7: cost of parts and quantities

4.2 Labor

People	Hourly Rate	Hours per Week	Total
Bilal Gabula	$20*2.5	24	14400
Osayanmo Osarenkhoe	$20*2.5	24	14400
Gerard McCann	$20*2.5	24	14400

Table 8: cost of labor

5. CONCLUSION

5.1 Accomplishments

The main goal of the project was to create a device to track the location of an animal using the GPS system and wirelessly transfer the data to a base station. The device should be implantable into an otter and also have a reasonable live span while accurately track otter movement patterns. Given the complexity of the assignment the accomplishments achieved during the semester were impressive. Ultimately a small device that records its current location. This device included a GPS antenna, GPS chip, microcontroller, memory chip, and switch. All of these components were chosen for their low power consumption and small size footprint as well as being integrated together on a 1 by 2 inch 4-layer PCB. The same PCB also included an RF chip, matching circuit and antenna, although this portion did not work as described in the RF system test and uncertainties section. The system is able to accurately store GPS coordinates and store them in nonvolatile memory.

The RF system was supposed to communicate with the base station which also has the capability to easily offload the data for the researchers. The only fully functioning portion of the base station is the microcontroller itself. The RF system on the base station is in the same state as that which is on the implant. The SD card hardware is completely set up as well as the file system interface design.

Given there were so many components of the project and the low power and size requirements the amount of goals achieved was admirable. All the hardware for the device is in place. We still need to write the rest of the code as well as additional debugging to ensure their functionality. The main data acquisition portion of the device is fully functioning including the software portion. Even though many of the design requirements necessary to have the device implanted into the otter were not fully met, in the scope of a senior design class the project should be considered a success. Every one of the group members learned numerous things about the design process and the work that goes into a fully functioning electronic device.

5.2 Uncertainties

As mentioned previously in the report there are subsystems of both the base station and the implant that are not fully functional. The SD card communication with the microcontroller has not been fully set up. This is a relatively standard SPI interface from a hardware perspective and the raw data transfer is akin to the mechanism used to communicate with the flash memory in the implant. This portion of the interface is essentially complete. The way the SD card handles different commands and stores the data is more complicated. It is possible that the group will finish the SD card subsystem of the project before the end of the semester, but at the time of writing it was not completed.

The RF communication link between the base station and the implant was also not completed. In this situation the communication between the microcontroller and the RF transceiver is fully functioning and tested but the actual RF portion of the circuit does not produce any output. The possible problems were discussed in the RF test section, but the debugging of the system is complicated and will likely not be completed in this semester.

5.3 Ethical considerations

During the design and implementation of the implant all the aspects of the IEEE code of ethics were followed. Since the device will be implanted into an animal there are extra issues that arise. We have to make sure the device does not bother or injure the otter. The area of greatest concern is the degradation of materials inside the otter. The product was designed so that it can last indefinitely inside the otter. A casing that do not degrade in a subcutaneous environment was found. Throughout the project data was correctly recorded and not altered, and while in lab any groups that asked for assistance were helped in the best way possible.

5.4 Future work

The current team plans to continue the project in the next semester as an independent study course. The goal of this will be to meet all the original design requirements as well as improve the size of the system and decrease its power requirements.

There are a number of feasible ways to reduce the size of the system. The easiest to implement would be changing the package type of the microcontroller. There are QFN(Quad Flat no Leads) versions of the Attiny 1634 that could easily replace our current SOP package. The microcontroller is currently the biggest component of the system and this would result in a significantly thinner final implant. Totally redesigning the implant components could implement many of the functionalities on a single chip. There is a TI CC430F5123 microcontroller RF transceiver built into a single IC that would significantly reduce the board size. The switch from an Atmel to a Texas Instruments chip might also remove the need for a separate memory chip as it is possible to store the location data into the flash memory. Looking ahead significant size reductions can be achieved and the initial goal of an implantable device is certainly a possibility.

REFERENCES

[1] SkyTraq Technology. Data sheet for Venus638FLPx GPS Receiver [Online]. Available : http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/GPS/Venus638FLPx.pdf

[2] Atmel. Datasheet for Attiny 1634 an 8-bit microcontroller with 16K Bytes In-System Programmable Flash. [Online]. Available: http://www.atmel.com/Images/doc8303.pdf

[3] Linear Technology. Data sheet for LTC4071 a Li-Ion/Polymer shunt battery charger system with low battery disconnect. [Online]. Available: http://cds.linear.com/docs/Datasheet/4071fc.pdf

[4] Linear Technology. Data sheet for LTC3588-1 a piezoelectric energy harvesting power supply. [Online]. Available: http://cds.linear.com/docs/Datasheet/35881fa.pdf

[5] Texas Instruments. Data sheet for CC1000 a single chip very low power RF transceiver. [Online]. Available: http://www.ti.com/lit/ds/symlink/cc1000.pdf

[6] Piezoelectric Systems. Data sheet for Energy Harvester Quick Mount 103. [Online]. Available: http://www.piezo.com/catalog8.pdf%20files/Cat8.43.pdf

[7] Sarantel. Data sheet for SL1024 (GeoHelix -M) a 2nd generation active helical GPS antenna. [Online]. Available: http://www.sparkfun.com/datasheets/GPS/SL1204%20Product%20Specification_v2_10_2009.pdf

Appendices

Appendix A

Requirement and Verification Table

Requirements	Reasoning	Testing
I. GPS System 1. Valid location (Two or Three dimensional location and time stamp) acquisition within 10 minutes of power up	1. The GPS system needs to acquire a valid signal within 10 minutes of power up so the GPSTimer does not overflow. . Chip needs to retrieve the Longitude, Latitude, time stamp and (optionally) Altitude for every properly stored GPS coordinate in designated time.	1. Power the GPS system by connecting 3.3V to pins 58 & 2 of the GPS chip and to the Vcc pin of the GPS antenna. Connect ground to pins 10 & 11 of the GPS chip and pins 1 & 3 of the antenna. Connect RX of a tested and ready µC to pin 44 and connect the antenna pin 2 to the GPS chip pin 32. Within 10 minutes of these connections, the µC should receive a $GPGGA (in ASCII) through the RX terminal followed by the location co-ordinates. Use the datasheet [2] to check if location is within 100m of actual location.
	1. a) Make sure the measurement environment has a signal by checking for GPS activity on a thoroughly tested device. 1 b) The GPS chip needs a signal of at least -148 dBm to acquire a location fix.	1. a) Look for a GPS signal using a smartphone with AGPS (Assisted Global Positioning System, i.e. turn off sensor abiding and WIFI) turned off. 1. b) Use a signal analyzer to measure the amplitude of the antenna output at the RF out pin of the antenna. 1. c) Check amplitude from another antenna (rerun Test I.1.a with another antenna)
2. Non directional	2. The antenna needs to be non-directional so as to receive the GPS satellite signal regardless of the orientation of the otter.	2. Check time to make GPS acquisitions in different antenna orientations using a stop watch. (Cold start every time, i.e. restart the GPS power before every test) 2.a) Check output amplitude from the antenna (Test I.1.a) using different orientations
	2.b) Varying current will result in varying acquisition times.	2. b) Ensure the power to GPS chip is not varying by using a current meter to measure the current into the chip at the VCC pins.
3. Low power consumption	3. To save power and ensure long product life, we need to make sure the	3. Use current meter to measure current from the battery during GPS acquisition

	GPS system does not consume excessive power.	when antenna is used in active mode. Power consumed (measured current * 3.3V) should be less than 264mW. (Use connections as in I.2.b)
	3. a) Ensure that the power saver mode on the GPS chip works	3. a) Use current meter to measure the current from the battery in normal mode and in power saver mode. Power saver mode consumption should be less than 75% of normal mode consumption.
	3. b) To check if the problem is in the software.	3. b) Hardwire power saver mode by connecting pin 8 of GPS chip to Vcc and rerun Test I.3.a.
4. Send data in Skytraq Binary format via UART (Instead of NMEA)	4. In order to minimize the data being transmitted without reducing data quality we will be transmitting and storing data in binary format. 4.a) Querying the software version is the simplest command that utilizes the TX and RX	4. Use tested LCD-µC pair with same connections as in Test I.1 to receiving location data. 4. a) Check the communication of GPS chip by querying the software version. Details of test given in GPS chip datasheet[5] 4.a.i) Check GPS chip power using a volt meter across its terminals
	4.b) To make sure there is no error in SkyTraq Binary conversion	4.b) Read data in NMEA using the µC
5. Greater than 100m accuracy	5. 100 m accuracy is reasonable for tracking animal movement patterns over long periods.	5. Use a USB serial convertor and connect the TX and RX terminals of the GPS chip to the RX and TX terminals of the serial convertor. Power the GPS chip and connect the grounds of the convertor and the GPS chip. Receive the co-ordinates using any standard serial reading program. Check received co-ordinate in Google Maps and compare with confirmed GPS location.
	5.a) Check that power saver mode is not the problem 5.b) Check that antenna orientation is not the problem	5. a) Rerun Test I.3 when not in power saver mode. 5. b) Rerun Test II.4 with a different antenna orientation.
II. Implant µC 1.Store 48 GPS data points (16 in 256 bytes in EEPROM) (32 in 512 bytes in SRAM)	1. µC should be able to store locations corresponding to ~10 days of activity. Each GPS location will include latitude (4 bytes), longitude (4 bytes), time stamp (Has to be reduced from 6 bytes to 4 bytes), and altitude (4	1. Using a GPS chip we will generate several GPS locations and store them on the chip (connections as in Test I.1) Using the LCD screen we will be able to see the multiple GPS locations. Also turn off power in between write and read operations. Use tested µC-LCD pair.

	bytes). Also verify non-volatility of the EEPROM 1.a) Check if the problem is in storage algorithm 1. b) Check if GPS system is sending data to storage.	1. a) Hard code data using test program and check values. 1. a.i) Check the storage code program for bugs. 1.b) Check GPS communication by rerunning Test I.1
2. UART interface should work at 4800 BAUD	2. The GPS chip communicates with the microcontroller at a minimum of 4800 baud using one of the UART interfaces.	2. Using a standard USB to serial convertor we will communicate with a computer to check that the UART works at 4800 BAUD. Connecting TX, RX and GND of the µC to the RX, TX and GND of the serial convertor respectively. 2. a) Check that the µC is powered using a voltmeter across its VCC and GND terminals.
3. Consumes less power during sleep mode than during run time.	3. The micro controller spends most of its time in sleep mode. This would help utilize as little power as necessary.	3. Using a current meter to measure current (into the VCC pin) test and make sure power in sleep mode is less than the power in active mode.
III. RF Chip 1. Communicate at a minimum distance of 10 meters	1. The otters are known to get within at least a 10 meter radius of a known location. Use two tested µCs to simulate the base station and implant, test the communication between two RF chips 10 meters apart. 1. a) Make sure the RF chip has power 1. b) Confirm that the RF chips are outputting data 1. c) Test RF Transmitter 1. d) Test RF Receiver	1. Connect the two RF chips to two different µC's as described in the schematic, and send a test signal from the transmitter to the receiver. Received data should be identical to the sent data. 1.a) Check power to the RF chips using a voltage meter connected to its VCC and GND pins 1. b) Connect the RF_OUT pin out to a signal analyzer. Output should be same as test data. 1. c) Using a signal analyzer 10m away with a wire antenna, analyze the transmitted signal. 1. d) Connect the output pins of the test receiver RF chip to a data analyzer when transmitter is within range and transmitting a test signal. Output simulation should be same as test data being sent from the tested transmitter

2. Lower power consumption in receive mode	2. 10 mA is a reasonable low power receive for sub 1Ghz RF.	1. e) If problems persist, debug µC-RF software interface. (including compiler optimization) 2. Use current meter to measure the current used by RF chip when in Receive mode. Measured value should be less than 10 mA.
3. Low power consumption in in transmit mode	3. This balances power consumption with communication distance and reliability without creating unreasonable expectations for a cheaper RF chip	3. Use current meter to measure the current used by RF chip when in Transmit mode. Measured value should be less than 17mA 3. a) Change Power output configuration till spec is met.
IV. Rechargeable Battery & Charging System 1. There should have a minimum of 60 mAh at 3.0-3.6V	1. 60 mAh estimated maximum daily power usage by the implant. The total power stored will be supplemented by the VEG.	1. Using a resistor and voltmeter hooked up to the positive and negative terminals of the battery, we will run down the battery to test the energy rating of the battery.
2. Able to provide a current of 60 mA for ten minutes.	2. Battery must be able to supply 60 mA of current continuously during active mode, which has a timer of ten minutes.	2. Using a resistor, voltmeter and current meter (similar to Test IV.1 except the current meter measure the battery current) to consume 60mA. The battery should be able to provide the required current for at least 10 minutes.
3. Able to charge the batteries with short instantaneous bursts of power.	3. Power generated by the piezoelectric crystals are in short instantaneous burst, thus the charging system must rectify and buffer these currents to properly charge the batteries.	3. Using a voltmeter and a function generator, we will measure the battery voltage while charging the battery up using short bursts of energy similar to those created by the charging system and verify that the battery charges as expected.
V. Vibrational Energy generator (VEG) 1. Generate minimum of 60 mAh per day	1. The GPS unit will be need 60mAh a day to have enough energy to function.	1. Using a half-wave rectifier and a capacitor set up we will measure the energy that one vibration would create. Using this value we will estimate the amount of energy the system will generate inside the otter.

2. Have dimensions of no more than 50mm x 4mm x 4mm	2. To keep to the size constraints of the overall implant, the VEG must not be larger than the listed dimensions.	2. We will use a ruler to measure the lengths of the largest sides of the VEG.
VI. Base Station Microcontroller 1. Minimum 3 kB non-volatile data storage. 2. Including RF chip and Antenna should be consuming less than 30 mA	1. The storage is for at least 4 month intervals of GPS data from 4 otters (3 kB) 2. We need the base station to run without needing to recharge for at least a week.	1. Rerun Test II.1 to with the base station and turn off the power to the µC in between the data write and data read. 2. Use a current meter test the current consumption.
VII. USB connector and interface 1. Connect, power and write files to USB as required	1. For ease of transfer of data. 1. a) To check if there errors in the USB format of the data being transferred.	1. Use the USB interface to write a sample GPS text file to a USB stick and check it on the computer. 1.a) Read the data directly from the µC to insure the data has actually been written
VIII. Large Battery 1. Greater than 5 V output power for 2 weeks	1. Assuming data is retrieved once every 1-2 weeks, the battery must maintain power to the RF chip for this time frame.	1. Test the battery capacity by running it down using a large resistor while measuring the voltage and current using a voltmeter and current meter. Similar to set up in Test IV.2

Size/Weight Requirements

The weight will be less than 1.5lb

The size will be: 15mm-20mm wide 20mm-25mm thick 95mm-100mm long

Casing Requirements

Casing must completely isolate the device from the otter and last at least 9 years.

Appendix B

To look at include files refer to Appendix E.

Test Code Files

GPS Test

```
/************************************************************
Project                                 : IROTS
File                                    : gps_com_check.c
Date                                    : 12/4/2012
Author                                  : Bilal
Chip type                               : ATtiny1634
Clock frequency                         : 8.0 MHz
This code should communicate with the GPS chip and
receive a GPS fix. The LED should start flashing
multiple times depending on the number of satellites
in view.
This code should aslo write the gps location to the
FRAM but that code has to be debugged.
************************************************************/
#define F_CPU 1000000UL
#include <avr\io.h>
#include <util\delay.h>
#include <avr\interrupt.h>
#include "usart.h"
#include "gps.h"
int main(void)
{
 DDRC=0xFF;//output
 PORTC=0xFF;
 PORTA=0x10;
 DDRA=0xEF;//output
 PORTA = 0x10;
 char a;
 char data;
 uint8_t s;
 gps_struct gps_temp;
 master_slave_init();
 gps_turn_on();//Turn on GPS
  PORTA |= (1<<6);//Turn on RF
 _delay_ms(50);
 TRX_init();
 ms_lcd_clrscr();
 data = receive_byte();
 while (data != 0x24)  data = receive_byte();
 gps_init();
 gps_pos(&gps_temp);
 store_gps(gps_temp);
 while(1)
                {
                gps_pos(&gps_temp);
                s = (0x0F)&&(gps_temp.sat);
                while(s>0)
                {
                                PINA=0x08;
                                _delay_ms(50);
                                PINA=0x08;
                                _delay_ms(50);
                                s--;
                }
                _delay_ms(30);
                PINA=0x08;
                _delay_ms(500);
}
}
```

Power Test

```
/*************************************************************
Project                                 : IROTS
File                                    : sleep_test.c
Date                                    : 12/04/2012
Author                                  : Bilal Gabula,
                                          Osayanmo R Osarenkhoe,
                                          Gerard McCan
Chip type                               : ATtiny1634
Clock frequency                         : 1.0 MHz
This file tests the different power consumption modes of the implant.
The modes are GPS on, ATtiny on and ATtiny asleep
This runs through only once to check the power and
the chip needs to be reset to re-run the test.
*************************************************************/
#include <avr\io.h>
#include <util\delay.h>
#include <avr\interrupt.h>
#include <avr\wdt.h>
#include <avr\sleep.h>
#include "usart.h"
#include "gps.h"

uint8_t time_count = 0;

void wdt_en(void)
{
 cli();//Disable golbal interrupts
 wdt_reset();//Reset wdt
 //Enable WDT interrupt and set prescaller to 8s
 WDTCSR |= (1<<WDIE)|(1<<WDP3)|(1<<WDP0);
 sei();//Enable global interrupts
}

void go_to_sleep(void)
{
 cli();
 MCUCR |= (1<<SM1);//Set sleep mode to power down
 MCUCR |= (1<<SE);//Enable sleep
 sei();
 sleep_cpu();
}

//Interrupt Vector for WDT.
ISR(WDT_vect)
{
 MCUCR &= ~(1<<SE);//Disable sleep
 WDTCSR |= (1<<WDIE);
 time_count++;
 if (time_count < 0x02)
  go_to_sleep();
}


int main(void)
{
 uint8_t i = 0;
 DDRC=0xFF;//output
 PORTB=0x00;
 DDRB=0xFF;//output
 PORTA=0x10;
 DDRA=0xEF;//output
 PINA |= 0x08;
 //Power test begin
 //high power level
 gps_turn_on();
 for(i = 0; i < 20; i++)
 {
   _delay_ms(500);
 }
 //blink three times
 for(i = 0; i < 3; i++)
 {
    _delay_ms(100);
                    PINA |= 0x08;
                    _delay_ms(100);
  PINA |= 0x08;

 }
 //medium power level
 gps_turn_off();

  for(i = 0; i < 20; i++)
 {
   _delay_ms(500);
 }
  //blink two times
```

```
 for(i = 0; i < 2; i++)
{
   _delay_ms(100);
                    PINA |= 0x08;
                    _delay_ms(100);
  PINA |= 0x08;
}
//low power mode
char a;
wdt_en();
go_to_sleep();
cli();
 //blink two times
 for(i = 0; i < 1; i++)
{
   _delay_ms(100);
                    PINA |= 0x08;
                    _delay_ms(100);
  PINA |= 0x08;
}
while(1)
{
                    _delay_ms(500);
                    PINA |= 0x08;
                    _delay_ms(500);
}
}
```

Appendix C

This Appendix shows the test boards used to test each individual component.

Microcontroller Breakout Board

This breakout board includes a reset switch, a test LED and a programming header. It was used to run all the tests on the other breakout boards as this was the microcontroller used for all the other tests.

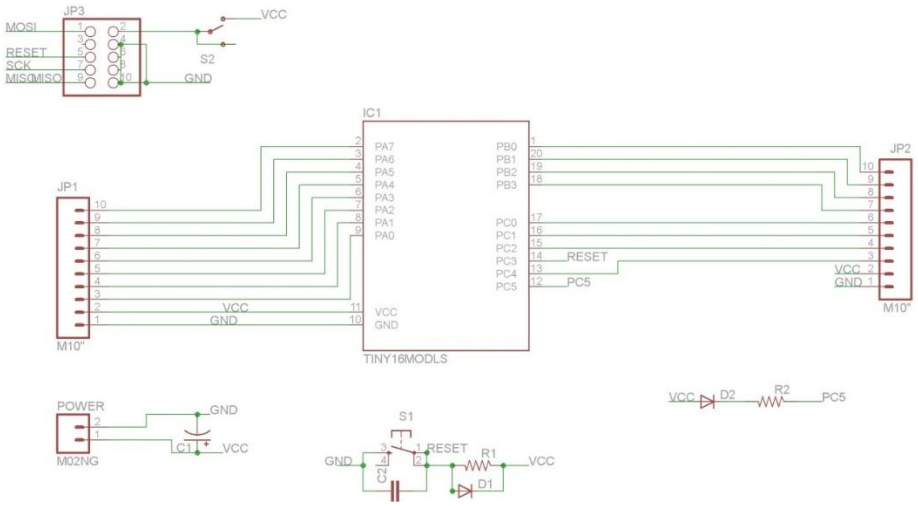


Figure 10: Attiny 1634 Breakout Schematic

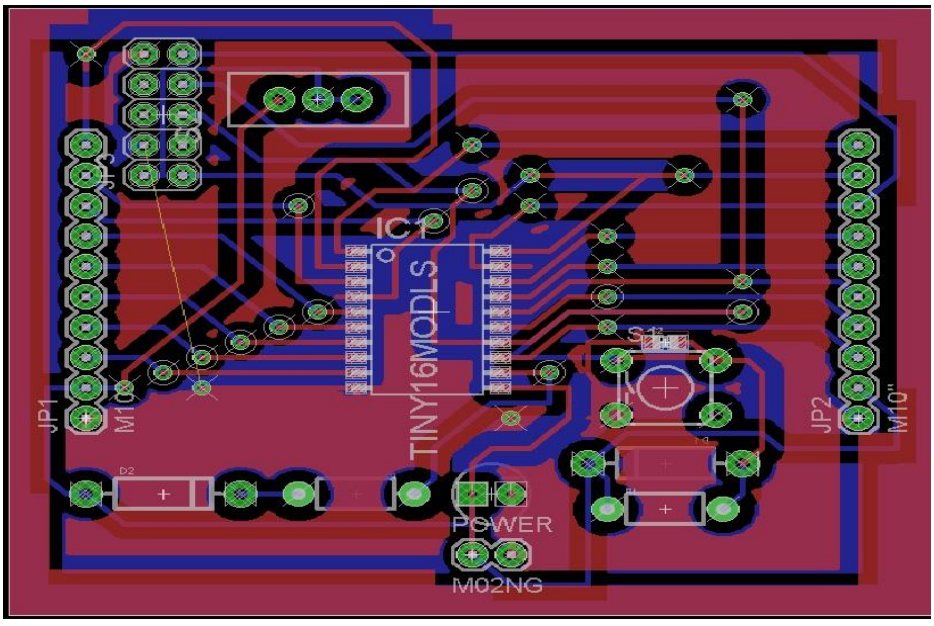


Figure 11: Attiny 1634 Breakout Board

RF Transceiver and Flash Chip Breakout Board

This was one of the more complicated breakout boards as it included the matching circuitry for the RF antenna and chip. This was used to test the communication with the RF chip (successful) it was used for RF transmission (unsuccessful), RF receiving (unsuccessful), Flash chip communication (successful) and Flash chip memory read and write (successful).

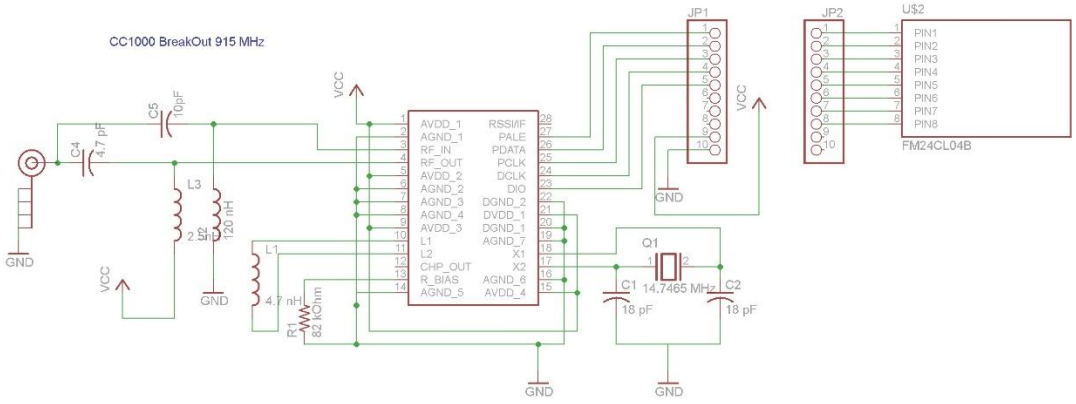


Figure 12: CC1000 Breakout Schematic

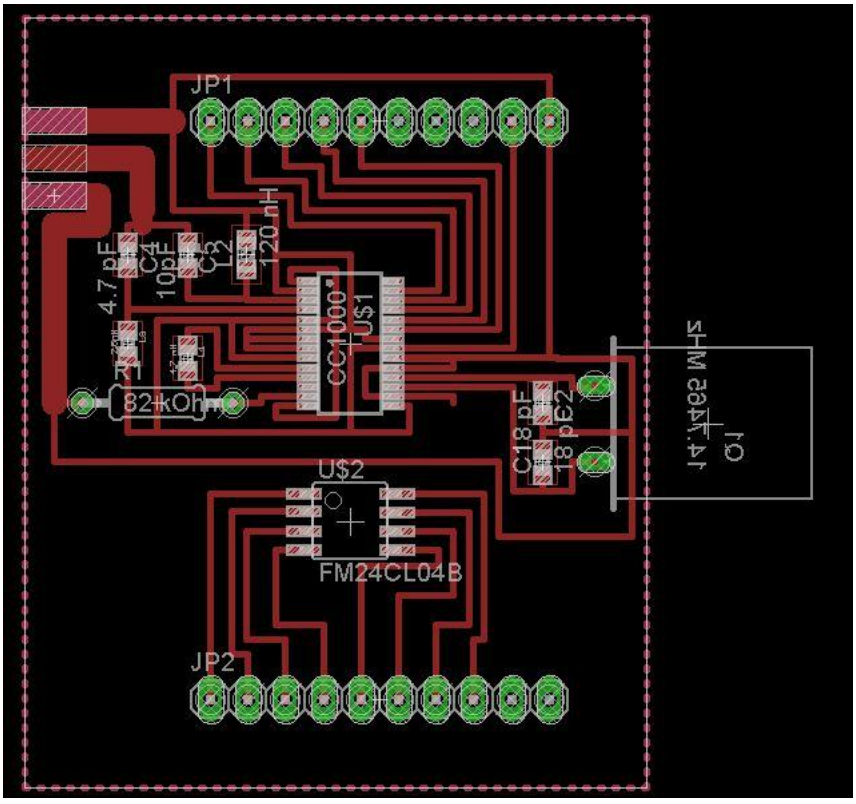


Figure 13: CC1000 Breakout Board Top Layer

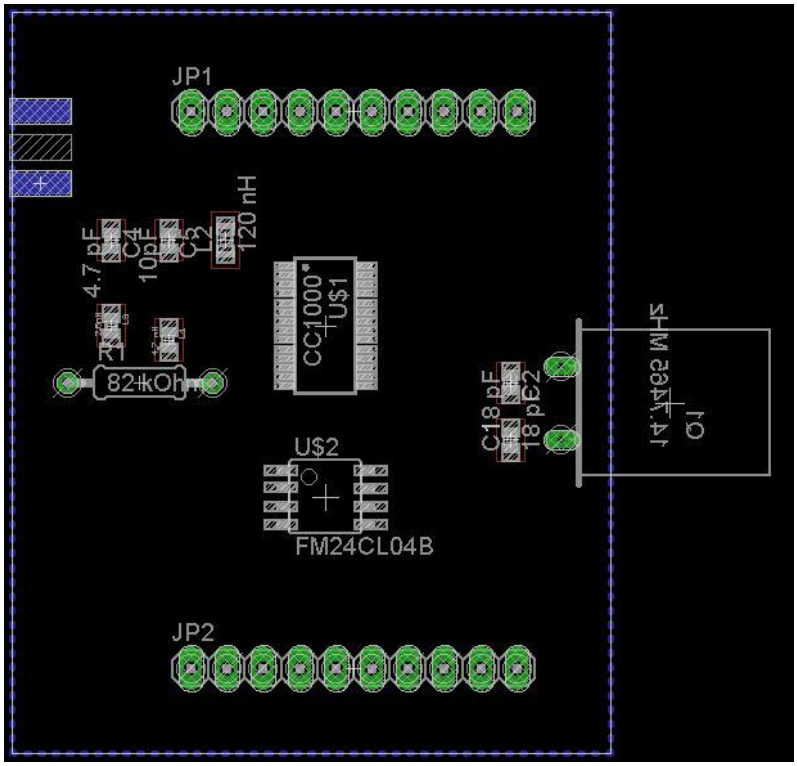


Figure 14: CC1000 Breakout Board Bottom Layer

Power Switch Breakout Board

This breakout board was to test the power switch used to turn on power to the GPS and RF sections of the implant. The test was simply to turn on the switch and measure voltage at the output depending on the current drawn. The test proved that the switch had only a drop of 50mV when drawing 100mA and therefore could be used to power the GPS chip.

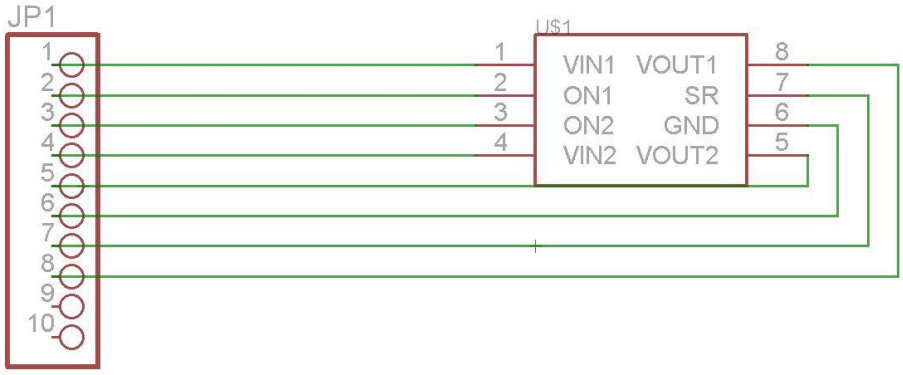


Figure 15: TI TPS Switch Schematic

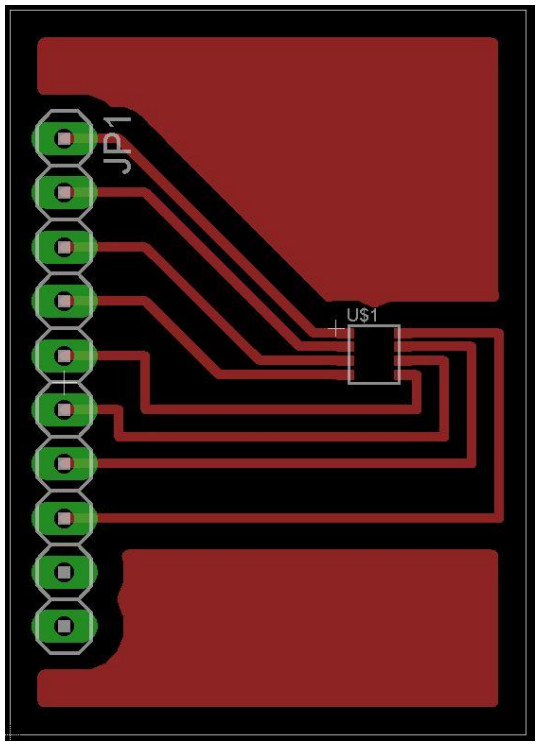


Figure 16: TI TPS Switch Board

Appendix D

Implant Layout

This layout is with respect to the implant schematic shown in the Schematic section.

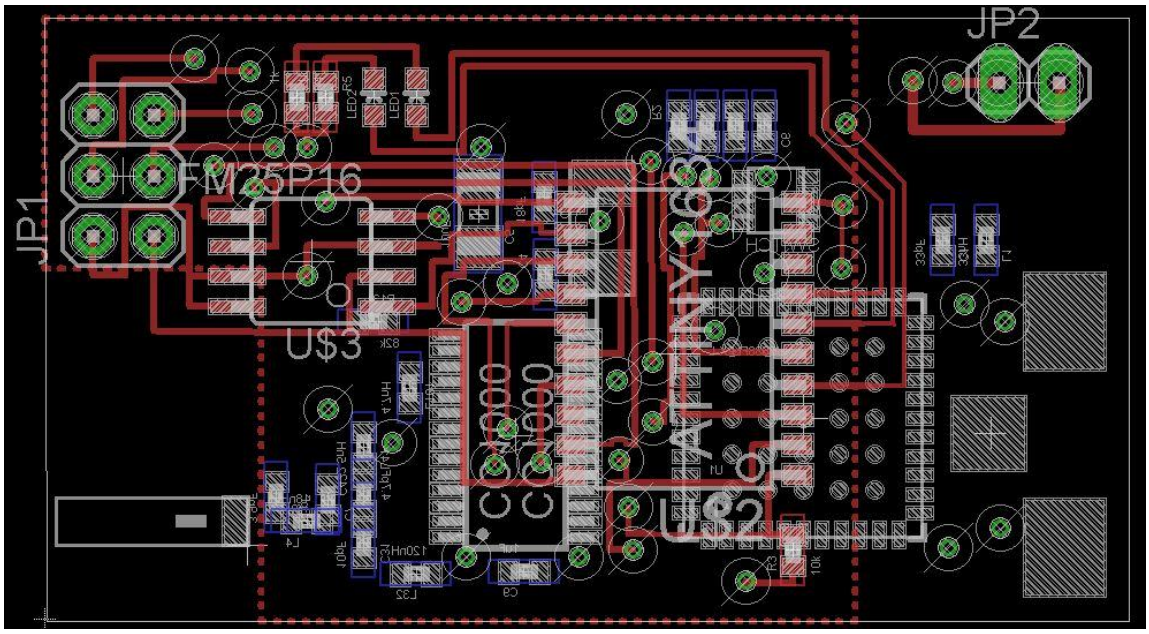


Figure 17: Final Implant Board Top Layer

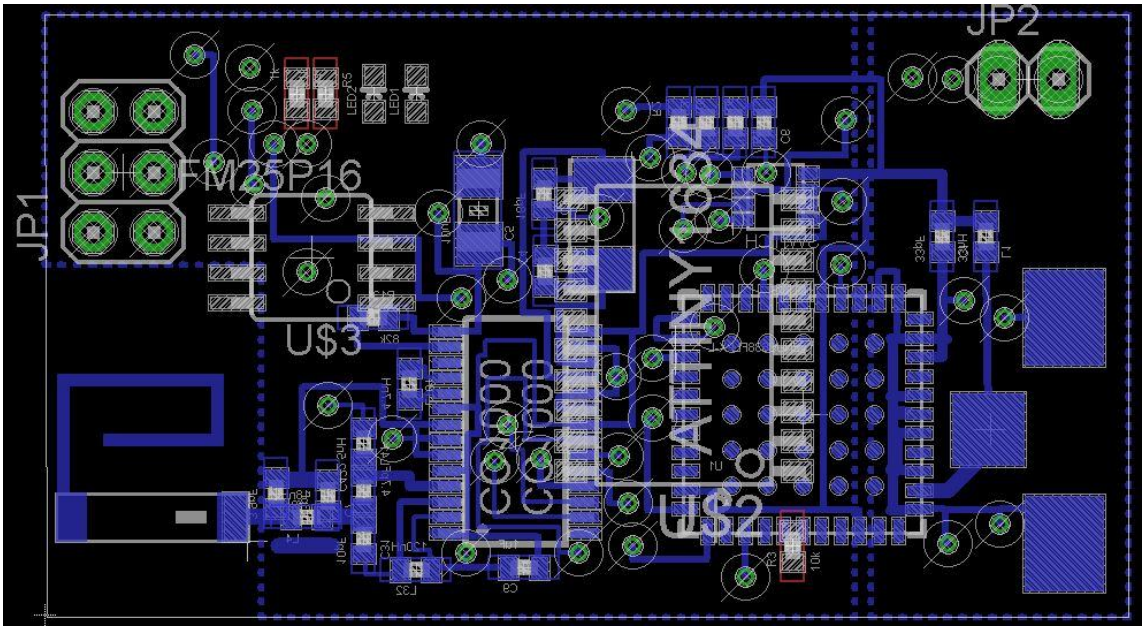


Figure 18: Final Board Bottom Layer

Base Station Layout

This layout is with respect to the base station schematic shown in the Schematic section.

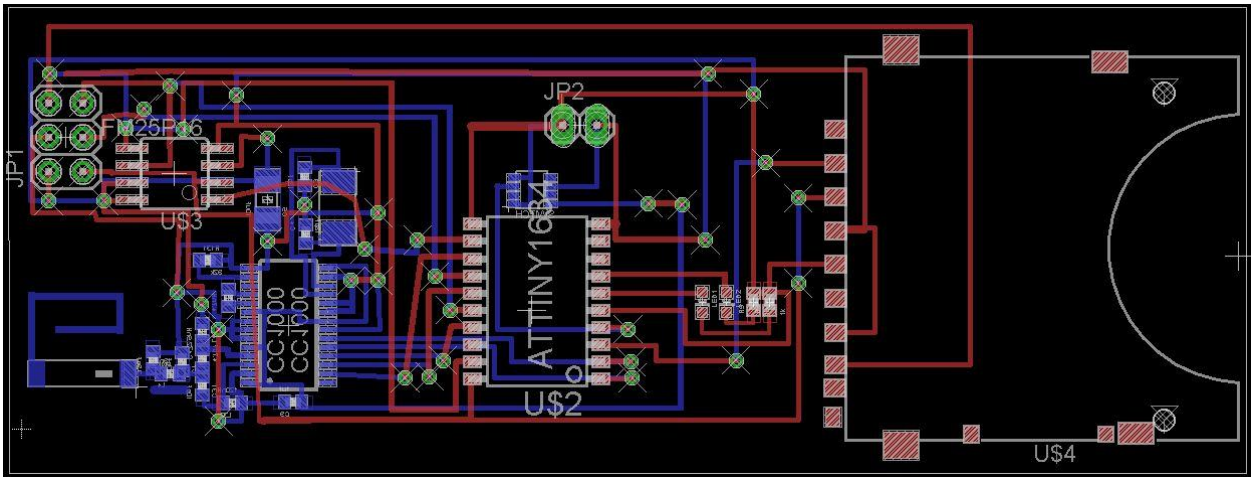


Figure 19: Base Station Board (Top and Bottom Layers Are Shown)

Appendix E

Include Files

GPS.H

```
/*********************************************************
File                                                    : GPS.H
This File includes various GPS functions.
It requires usart.h to be included to work.
There are also some functions that store data to an
external flash chip.
*********************************************************/
#ifndef gps_h
#define gps_h
#define PGPS_SEL  5
#define PGPS_PORT PORTA

typedef struct gps_struct
{
 uint8_t sat;
 uint32_t lat;
 uint32_t lon;
 uint16_t wkn;
 uint8_t tow;
}gps_struct;

static void gps_get_ak(void)
{
 uint8_t data;
 data = receive_byte();
 while (data!=0xA0) data = receive_byte();
}

void gps_turn_on(void)
{
 PGPS_PORT |=  (1<<PGPS_SEL);
 _delay_ms(50);
}

void gps_turn_off(void)
{
  PGPS_PORT &=  ~(1<<PGPS_SEL);
 _delay_ms(50);
}

void gps_get_msg(void)
{
 uint8_t data, pl_m, pl_l;
 uint16_t pl;
 data = receive_byte();
 while (data!=0xA0) data = receive_byte();
 data = receive_byte();
 if(data == 0xA1)
 {
  pl_m = receive_byte();
  pl_l = receive_byte();
  pl=(pl_m<<8)|(pl_l);
  while(pl>0)
  {
   pl--;
   data = receive_byte();
  }
 }
}


void gps_version(void)
{

 send_byte(0xA0);
 send_byte(0xA1);
 send_byte(0x00);
 send_byte(0x02);
 send_byte(0x02);
 send_byte(0x00);
 send_byte(0x02);
 send_byte(0x0D);
 send_byte(0x0A);
 gps_get_ak();
}

/* sends the gps messages in the binary chip format */
static void gps_message(uint16_t p_length, char* payload)
```

```
{
  uint8_t checksum = 0;
  uint16_t i = 0;
  uint8_t first8_PL;
  uint8_t second8_PL;
  uint16_t pl_mask = 0x00FF;


  /*separate PL into two bytes*/
  first8_PL = (p_length >> 8);
  second8_PL = (p_length & pl_mask);
  while(i<p_length)
  {
   checksum ^= payload[i]; //calculating checksum
                    i++;
  }
  i = 0;
  send_byte(0xA0);
  send_byte(0xA1); //starting bytes
  send_byte(first8_PL); //payload length
  send_byte(second8_PL);
  //calculates checksum and sends payload
  while(i < p_length)
  {
    send_byte(payload[i]);
    i++;
  }
  send_byte(checksum);
  send_byte(0x0D);
  send_byte(0x0A);
}

void gps_power(void)
{
 char payload[]= {0x0c,0x01,0x01};//set to power saver mode and to update flash
 gps_message(0x03,payload);
 gps_get_msg();
 gps_get_msg();
}

void gps_nav_rate(void)
{
 char payload[]= {0x11,0x01,0x01};//set to one sec and to update flash
 gps_message(0x03,payload);
 gps_get_msg();
 gps_get_msg();
}

static void gps_get_trimmed_msg(uint8_t msg_id, uint16_t start, uint16_t end)
{
 uint8_t data;
 uint16_t i=0x1;
 data = receive_byte();
 while (data!=0xA0) data = receive_byte();
 data = receive_byte();
 if(data == 0xA1)
 {
  data = receive_byte();//pl_l
  data = receive_byte();//pl_u
  data = receive_byte();//msg_id
  if(data == msg_id)
  {
   while(i<end)
   {
    i++;
    data = receive_byte();
   }
  }
 }
}

void gps_pos_old(void)
{
 gps_get_trimmed_msg(0xA8,0xA,0x11);
}

void gps_time(void)
{
 gps_get_trimmed_msg(0xA8,0x4,0x9);
}

void gps_msg_type()//Change output message type
{
char payload[]= {0x09,0x02};
                                                    //msgid, set to output
 gps_message(0x02,payload);
  gps_get_msg();
  gps_get_msg();
```

```
}

void gps_sat_num(void)
{
 gps_get_trimmed_msg(0xA8,0x3,0x3);
}

void gps_init()
{
 gps_msg_type();
 gps_power();
 gps_nav_rate();
}

void gps_disable()
{
 char payload[]= {0x11,0x00,0x01};//set to 0 sec and to update flash
 gps_message(0x03,payload);
 gps_get_msg();
 gps_get_msg();
}

uint8_t gps_pos(gps_struct * gps)
{
 uint8_t        fix = 0x00;
 uint8_t        sv = 0x00;
 uint8_t        data = 0x00;
 uint16_t       data_16 = 0x0000;
 uint16_t       wn = 0x0000;
 uint32_t       data_32 = 0x00000000;
 uint32_t       toww = 0x00000000;
 uint32_t       latt = 0x00000000;
 uint32_t       lonn = 0x00000000;
 data = receive_byte();
 while (data!=0xA0) data = receive_byte();
 data = receive_byte();
 if(data == 0xA1)
 {
  data = receive_byte();//pl_l
  data = receive_byte();//pl_u
  data = receive_byte();//msg_id
  if(data == 0xA8)
  {
                fix = receive_byte();//fix mode
                sv = receive_byte();//# sat in fix
                data_16 = receive_byte();
                wn |= (data_16<<8);
                wn |= receive_byte();//week #
                data_32 = receive_byte();
                toww |= (data_32<<24);
                data_32 = receive_byte();
                toww |= (data_32<<16);
                data_32 = receive_byte();
                toww |= (data_32<<8);
                data_32 = receive_byte();
                toww |= (data_32<<0);//Time of week
                data_32 = receive_byte();
                latt |= (data_32<<24);
                data_32 = receive_byte();
                latt |= (data_32<<16);
                data_32 = receive_byte();
                latt |= (data_32<<8);
                data_32 = receive_byte();
                latt |= (data_32<<0);//Lattitude
                data_32 = receive_byte();
                lonn |= (data_32<<24);
                data_32 = receive_byte();
                lonn |= (data_32<<16);
                data_32 = receive_byte();
                lonn |= (data_32<<8);
                data_32 = receive_byte();
                lonn |= (data_32<<0);//Longitude
  }
 }
 gps->sat = (fix<<6)|(sv);
 gps->wkn = wn;
 gps->lat = latt;
 gps->lon = lonn;
 gps->tow = (uint8_t)(toww>>18);
 if (fix>0)
 {
  store_gps(*gps);
  return 1;
 }
 else return 0;
}

void ms_disp_gps(gps_struct gps)
```

```
{
 ms_send_byte(0xA0);//Sync
 ms_send_byte(0xE2);//Command to disp pos
 ms_send_byte(gps.sat);//Fix mode and # Sattelites
 ms_send_byte(gps.tow);//Time of Week
 ms_send_byte(((uint8_t)(gps.wkn>>8)));//Week Number
 ms_send_byte(((uint8_t)(gps.wkn>>0)));
 ms_send_byte(((uint8_t)(gps.lat>>24)));
 ms_send_byte(((uint8_t)(gps.lat>>16)));
 ms_send_byte(((uint8_t)(gps.lat>>8)));//Lattitude
 ms_send_byte(((uint8_t)(gps.lat>>0)));
 ms_send_byte(((uint8_t)(gps.lon>>24)));
 ms_send_byte(((uint8_t)(gps.lon>>16)));
 ms_send_byte(((uint8_t)(gps.lon>>8)));
 ms_send_byte(((uint8_t)(gps.lon>>0)));//Longitude
}


void store_gps(gps_struct gps)
{
 uint16_t add;
 SPI_init();
 add = (flash_read(MEM_END)<<8)|(flash_read(MEM_END+1));
 if (add <= 0xFF00)
 {
 add++;
 flash_write(add,gps.sat);//Fix mode and # Sattelites
 add++;
 flash_write(add,gps.tow);//Time of Week
 add++;
 flash_write(add,((uint8_t)(gps.wkn>>8)));//Week Number
 add++;
 flash_write(add,((uint8_t)(gps.wkn>>0)));
 add++;
 flash_write(add,((uint8_t)(gps.lat>>24)));
 add++;
 flash_write(add,((uint8_t)(gps.lat>>16)));
 add++;
 flash_write(add,((uint8_t)(gps.lat>>8)));//Lattitude
 add++;
 flash_write(add,((uint8_t)(gps.lat>>0)));
 add++;
 flash_write(add,((uint8_t)(gps.lon>>24)));
 add++;
 flash_write(add,((uint8_t)(gps.lon>>16)));
 add++;
 flash_write(add,((uint8_t)(gps.lon>>8)));
 add++;
 flash_write(add,((uint8_t)(gps.lon>>0)));//Longitude
 flash_write(MEM_END,((uint8_t)(add>>8)));
 flash_write((MEM_END+1),(uint8_t)(add));
 master_slave_init();
 }
}

//This was renamed from read_gps_strcut
gps_struct read_gps_struct(uint16_t add)
{
 gps_struct gps;
 uint32_t temp;
 gps.sat =  flash_read(add);
 add++;
 gps.tow =  flash_read(add);
 add++;
 gps.wkn |= (flash_read(add)<<8);
 add++;
 gps.wkn |= (flash_read(add)<<0);
 add++;
 temp =  (flash_read(add));
 gps.lat |= temp<<24;
 add++;
 temp =  (flash_read(add));
 gps.lat |= temp<<16;
 add++;
 temp =  (flash_read(add));
 gps.lat |= temp<<8;
 add++;
 temp =  (flash_read(add));
 gps.lat |= temp<<0;
 add++;
 temp =  (flash_read(add));
 gps.lon |= temp<<24;
 add++;
 temp =  (flash_read(add));
 gps.lon |= temp<<16;
 add++;
 temp =  (flash_read(add));
 gps.lon |= temp<<8;
```

```
 add++;
 temp =  (flash_read(add));
 gps.lon |= temp<<0;
}

void send_all_gps_data(void)
{
 uint16_t curr_add = (flash_read(MEM_START)<<8)|(flash_read(MEM_START+1));;
 uint16_t end_add = (flash_read(MEM_END)<<8)|(flash_read(MEM_END+1)) ;
 curr_add++;
 while(curr_add<end_add);
 {
                ms_disp_gps(read_gps_struct(curr_add));//rename gps_strcut
                curr_add += 12;
 }
}
#endif
```

USART.H

```
/*****************************************************
File                                    : USART.H

This is the include file to use the USART protocoll
on the ATtiny1634
It includes SPI protocoll functions and funcrions
to read and write from an external FRAM memory
It has functions to communicate with a computer or a
slave microcontroller using USART
*****************************************************/
#ifndef usart_h
#define usart_h
#define XCKN1_DDR           DDRC
#define XCK1                1
#define PORT_CS                         PORTC
#define CS                              5
#define MEM_START   0x0000
#define MEM_END                         0x0002

void TRX_init(void)//initialize both TX and RX
{
        UBRR0H = (unsigned char)(12>>8);
        UBRR0L = (unsigned char)12;//setting the BAUD to 4800 for F_CPU to 1Mhz
        UCSR0B|= (1<<RXEN0)|(1<<TXEN0);//Enable TX and RX
        UCSR0C|= (3<<UCSZ00);//writing to UCSRC, async mode, parity disabled, 1 bit stop code, 8bit word.
}

void send_byte(uint8_t data)
{
        while(!(UCSR0A & (1<<UDRE0))){}//wait for the shift register to empty
        UDR0 = data;//put value inside the register
}


uint8_t receive_byte(void)
{
        while(!(UCSR0A & (1<<RXC0)));//wait for data to get received
        return UDR0;//return data from buffer
}

void SPI_init(void)//Initialize TX1, RX1 and XCK1
{
        UBRR1                   = 0;
        XCKN1_DDR  |= (1<<XCK1);//Setting Xck1 port pin to op, enables master mode.
        UCSR1C          |= (1<<UMSEL11)|(1<<UMSEL10);//Set MSPI mode of operation and set SPI data mode to 0.
        UCSR1C          &= ~((1<<0)|(1<<1)|(1<<2));
        UCSR1B          |= (1<<RXEN1)|(1<<TXEN1);//Enable Tx and Rx
        UBRR1                   = 12;                                                   //Set Baudrate to 4800
        PORT_CS |= (1<<CS);
}

uint8_t SPI_receive(uint8_t data)
{
        uint8_t to_return;
        PORT_CS &= ~(1<<CS);
        while(!(UCSR1A & (1<<UDRE1)));//wait for empty transmitt buffer.
        UDR1 = data;                                            //put data in buffer.
        while(!(UCSR1A & (1<<RXC1)));
        to_return = UDR1;
        while(!(UCSR1A & (1<<UDRE1)));
        UDR1 = 0xff;
        while(!(UCSR1A & (1<<RXC1))); //wait for data to be received
        to_return = UDR1;                                               //return data.
        PORT_CS |= (1<<CS);
        return to_return;
}

void Flash_WE(void)
{
        SPI_receive(0x06);
}

void SPI_send_word(uint8_t msb_data, uint8_t lsb_data)
{
        uint8_t dummy;
        PORT_CS &= ~(1<<CS);
        while(!(UCSR1A & (1<<UDRE1)));//wait for empty transmitt buffer.
        UDR1 = msb_data;                                        //put data in buffer.
        while(!(UCSR1A & (1<<RXC1)));
        dummy = UDR1;
        while(!(UCSR1A & (1<<UDRE1)));//wait for empty transmitt buffer.
        UDR1 = lsb_data;                                        //put data in buffer.
        while(!(UCSR1A & (1<<RXC1)));
        dummy = UDR1;
        PORT_CS |= (1<<CS);
}
```

```
void flash_write(uint16_t add, uint8_t data)
{
		Flash_WE();
		uint8_t dummy;
		PORT_CS &= ~(1<<CS);
		while(!(UCSR1A & (1<<UDRE1)));//wait for empty transmitt buffer.
		UDR1 = 0x02;                                                    //Opcode for mem write
		while(!(UCSR1A & (1<<RXC1)));
		dummy = UDR1;
		while(!(UCSR1A & (1<<UDRE1)));//wait for empty transmitt buffer.
		UDR1 = (uint8_t)(add>>8);       //Send msb Address
		while(!(UCSR1A & (1<<RXC1)));
		dummy = UDR1;
		while(!(UCSR1A & (1<<UDRE1)));//wait for empty transmitt buffer.
		UDR1 = (uint8_t)(add);                          //Send lsb Address
		while(!(UCSR1A & (1<<RXC1)));
		dummy = UDR1;
		while(!(UCSR1A & (1<<UDRE1)));//wait for empty transmitt buffer.
		UDR1 = data;                                                    //Send data
		while(!(UCSR1A & (1<<RXC1)));
		dummy = UDR1;
		PORT_CS |= (1<<CS);
}

uint8_t flash_read(uint16_t add)
{
		uint8_t dummy,to_return;
		PORT_CS &= ~(1<<CS);
		while(!(UCSR1A & (1<<UDRE1)));//wait for empty transmitt buffer.
		UDR1 = 0x03;                                                    //Opcode for mem read
		while(!(UCSR1A & (1<<RXC1)));
		dummy = UDR1;
		while(!(UCSR1A & (1<<UDRE1)));//wait for empty transmitt buffer.
		UDR1 = (uint8_t)(add>>8);       //Send msb Address
		while(!(UCSR1A & (1<<RXC1)));
		dummy = UDR1;
		while(!(UCSR1A & (1<<UDRE1)));//wait for empty transmitt buffer.
		UDR1 = (uint8_t)(add);                          //Send lsb Address
		while(!(UCSR1A & (1<<RXC1)));
		dummy = UDR1;
		while(!(UCSR1A & (1<<UDRE1)));
		UDR1 = 0xff;
		while(!(UCSR1A & (1<<RXC1))); //wait for data to be received
		to_return = UDR1;                                                       //return data.
		PORT_CS |= (1<<CS);
		return to_return;
}

void master_slave_init(void)//initialize both TX and RX for master slave mode
{
		UBRR1H = (unsigned char)(12>>8);
		UBRR1L = (unsigned char)12;//setting the BAUD to 4800 for F_CPU to 1Mhz
		UCSR1B|= (1<<RXEN1)|(1<<TXEN1);//Enable TX and RX
		UCSR1C|= (3<<UCSZ10);//writing to UCSRC, async mode, parity disabled, 1 bit stop code, 8bit word.
		UCSR1C&= ~(1<<UMSEL11);
		UCSR1C&= ~(1<<UMSEL10);
}


void ms_send_byte(uint8_t data)
{
		while(!(UCSR1A & (1<<UDRE1))){}//wait for the shift register to empty
		UDR1 = data;//put value inside the register
}


uint8_t ms_receive_byte(void)
{
		while(!(UCSR1A & (1<<RXC1)));//wait for data to get received
		return UDR1;//return data from buffer
}

void ms_disp_hex(uint8_t data)
{
 ms_send_byte(0xA0);//Sync
 ms_send_byte(0x8D);//Command to disp
 ms_send_byte(data);//send data
}

void ms_lcd_clrscr()
{
 ms_send_byte(0xA0);//Sync
 ms_send_byte(0x18);//Command to disp
}

void ms_lcd_putc(uint8_t data)
{
```

```
 ms_send_byte(0xA0);//Sync
 ms_send_byte(0x24);//Command to disp
 ms_send_byte(data);//send data
}

void ms_lcd_gotoxy(uint8_t x, uint8_t y)
{
 ms_send_byte(0xA0);//Sync
 ms_send_byte(0x41);//Command to disp
 ms_send_byte(x);//send x
 ms_send_byte(y);//send y
}


#endif
```

CC1000_Data_Transfer.H

```
/*****************************************************
Project                         : IROTS
Date                            : 11/5/2012
Author                          : Osayanmo R Osarenkhoe and Gerard McCann
Description     : This is the C code for recieving data from the RF chip
*****************************************************/
#ifndef CC1000_DATA_TRANSFER_H
#define CC1000_DATA_TRANSFER_H

#define PDATA                   2
#define PALE   4
#define PCLK   0
#define PORT_RF                 PORTC
#define PIN_RF                  PINC
#define DDR_RF                  DDRC
#define PRF_SEL 6
#define PRF_PORT PORTA

void init_CC1000(void);

uint8_t receive_data(void);

void send_address(uint8_t address, char R_Wb);

void send_data(uint8_t data);

void send_RF_data(uint8_t data);

#endif
```

CC1000_Data_Transfer.C

```
/*****************************************************
Project                         : IROTS
Date                            : 11/5/2012
Author                          : Osayanmo R Osarenkhoe and Gerard McCann
Description     : This is the C code for recieving data from the RF chip
*****************************************************/

#include "CC1000_Data_Transfer.h"
#include <util\delay.h>
#include <usart.h>


void rf_turn_on(void)
{
 PRF_PORT |= (1<<PRF_SEL);
 _delay_ms(50);
}

void rf_turn_off(void)
{
 PRF_PORT &= ~(1<<PRF_SEL);
 _delay_ms(50);
}

void send_address(uint8_t address, char R_Wb)
{
  uint8_t i = 0; //for loop counter
  address = address<<1;//Since the address is 7 bits long.
  /*initialize output pins*/
  PORT_RF |= (1<<PDATA)|(1<<PALE)|(1<<PCLK); //Set all output pins to 1
  DDR_RF |= (1<<PDATA)|(1<<PALE)|(1<<PCLK);
  //_NOP();
  _delay_us(10);


  PORT_RF &= ~(1<<PALE);    //PALE = 0
  _delay_us(10);

  /*send the 7 address bits*/
  for(i = 7; i > 0; i--)
  {
    if(address & ((uint8_t)(1<<i)))
      PORT_RF |= (1<<PDATA);
    else
      PORT_RF &= ~(1<<PDATA);
    _delay_us(10);
    PORT_RF &= ~(1<<PCLK);
    _delay_us(10);
    PORT_RF |= (1<<PCLK);

  }
  if((R_Wb == 'R')||(R_Wb == 'r'))
  {
                  PORT_RF &= ~(1<<PDATA);
```

```
                _delay_us(10);
                PORT_RF &= ~(1<<PCLK);
                _delay_us(10);
                PORT_RF |= (1<<PCLK);
                }
  else if((R_Wb == 'W')||(R_Wb == 'w'))
                {
                PORT_RF |= (1<<PDATA);
                _delay_us(10);
                PORT_RF &= ~(1<<PCLK);
                _delay_us(10);
                PORT_RF |= (1<<PCLK);
                }
                _delay_us(10);
                PORT_RF |= (1<<PALE);         //PALE = 1

}

inline uint8_t receive_data(void)
{
                uint8_t data_out = 0x00; //the data that is recieved from the rf chip
                uint8_t i = 0;                                          //for loop counter

                //convert the pdata pin to an input pin
                DDR_RF &= ~(1<<PDATA); //DDR_RF PDATA input
                _delay_us(10);
                PORT_RF |= (1<<PALE); //PALE = 1
                _delay_us(10);

                //read in data bit by bit
                for(i = 8; i > 0; i--)
                {
                                _delay_us(10);
                                PORT_RF &= ~(1<<PCLK);
                                if(PIN_RF & (1<<PDATA))
                                {
                                                data_out |= (1<<(i-1));
                                }
                                _delay_us(10);
                                PORT_RF |= (1<<PCLK);
                                _delay_us(10);
                }
                return data_out;
}

inline void send_data(uint8_t data)
{
  uint8_t i = 0; //loop counter

                PORT_RF |= (1<<PALE); //PALE = 1
  _delay_us(10);

  //pass in data bit by bit
  for(i = 8; i > 0; i--)
  {
                                if(data & (1<<(i-1))) PORT_RF |= (1<<PDATA);
                                else PORT_RF &= ~(1<<PDATA);
                                _delay_us(10);
                                PORT_RF &= ~(1<<PCLK);
                                _delay_us(10);
                                PORT_RF |= (1<<PCLK);
                                _delay_us(10);
  }
}

/*Need to talk to osa to see how to initialize*/
inline void init_CC1000(void)
{
}

/*Sends one byte to the CC1000 using transparent UART */
inline void send_RF_data(uint8_t data)
{
  send_byte(data);
}

/*Receives one byte from CC1000 using transparent UART */
inline uint8_t receive_RF_data(void)
{
  return receive_byte();
}

void cc1k_send_comm(uint8_t address, uint8_t data)
{
 send_address(address, 'W');
 send_data(data);
}
```

```
uint8_t cc1k_get_comm(uint8_t address)
{
  send_address(address, 'R');
  return receive_data();
}
```

Appendix F

Bugs List

//Bug List
Clock output to be measured. Done FCPU is now at 1.05MHz
Rx pin of GPS breakout is messed up. have to re-solder. Done
Some pins on the breakout of the bugstrip have come out. Check connections. Done
3.3V regulator being used goes into protection mode if GPS is hooked up at the output. Unplug the GPS to get the regulator started. Then re-plug it in.
Reset pin isn't pulled high on the implant. Easy fix using connectors. Done
VCC and Vtest are not the same netlist. Will need to hardwire that somehow. Done.
P_GPS and P_RF named wrongly on the schematic rename. Done.
P_GPS and GND shorted on implant check. Re-soldered and Fixed.
Check if microcontrollers can be recovered.
Check if GPS is the problem in communication with RF chip.
Debug GPS structure storing file system.

Appendix G

Implant Image

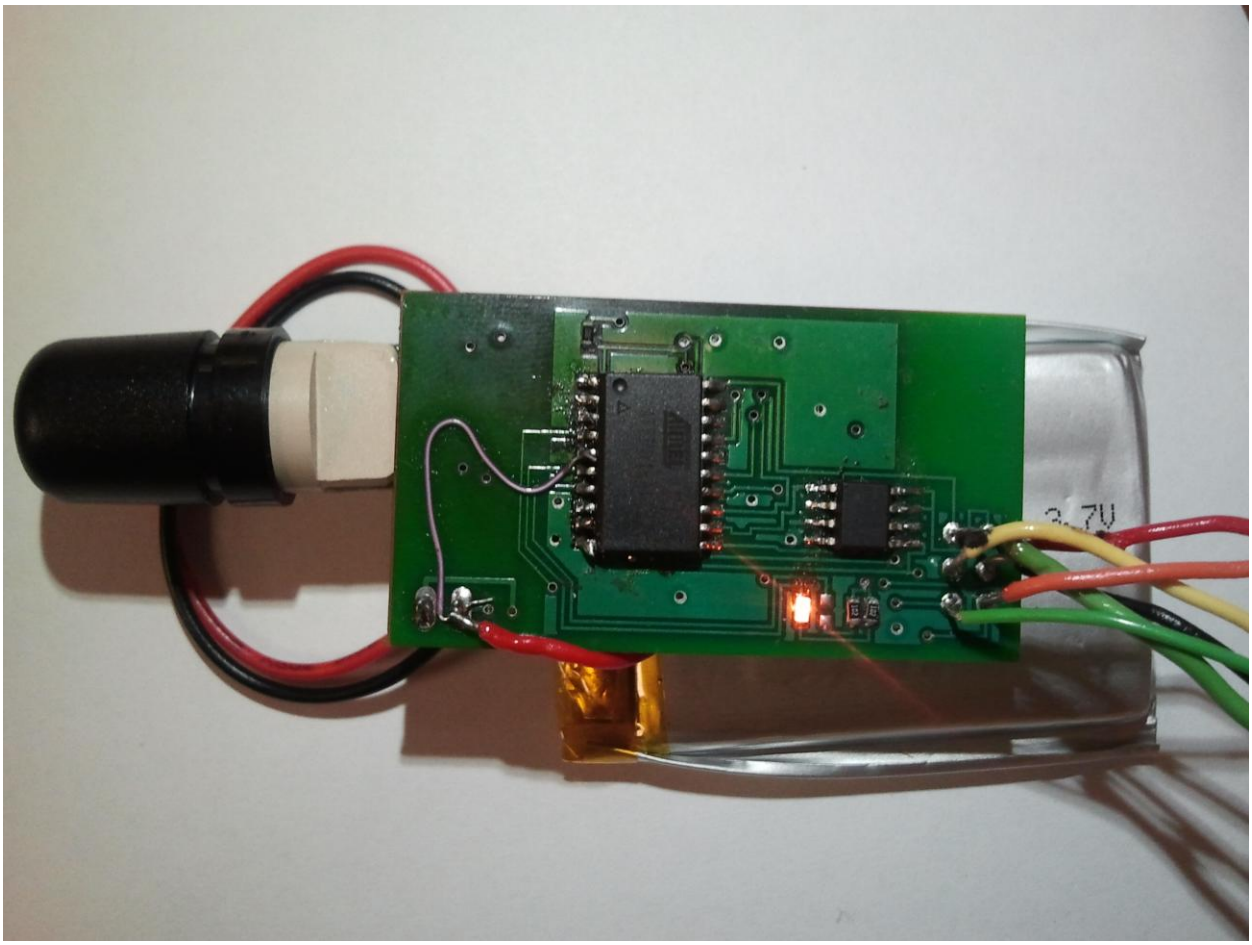


Figure 20: Final Implant Board

Figure 21: Implant Battery Image

Base Station Image

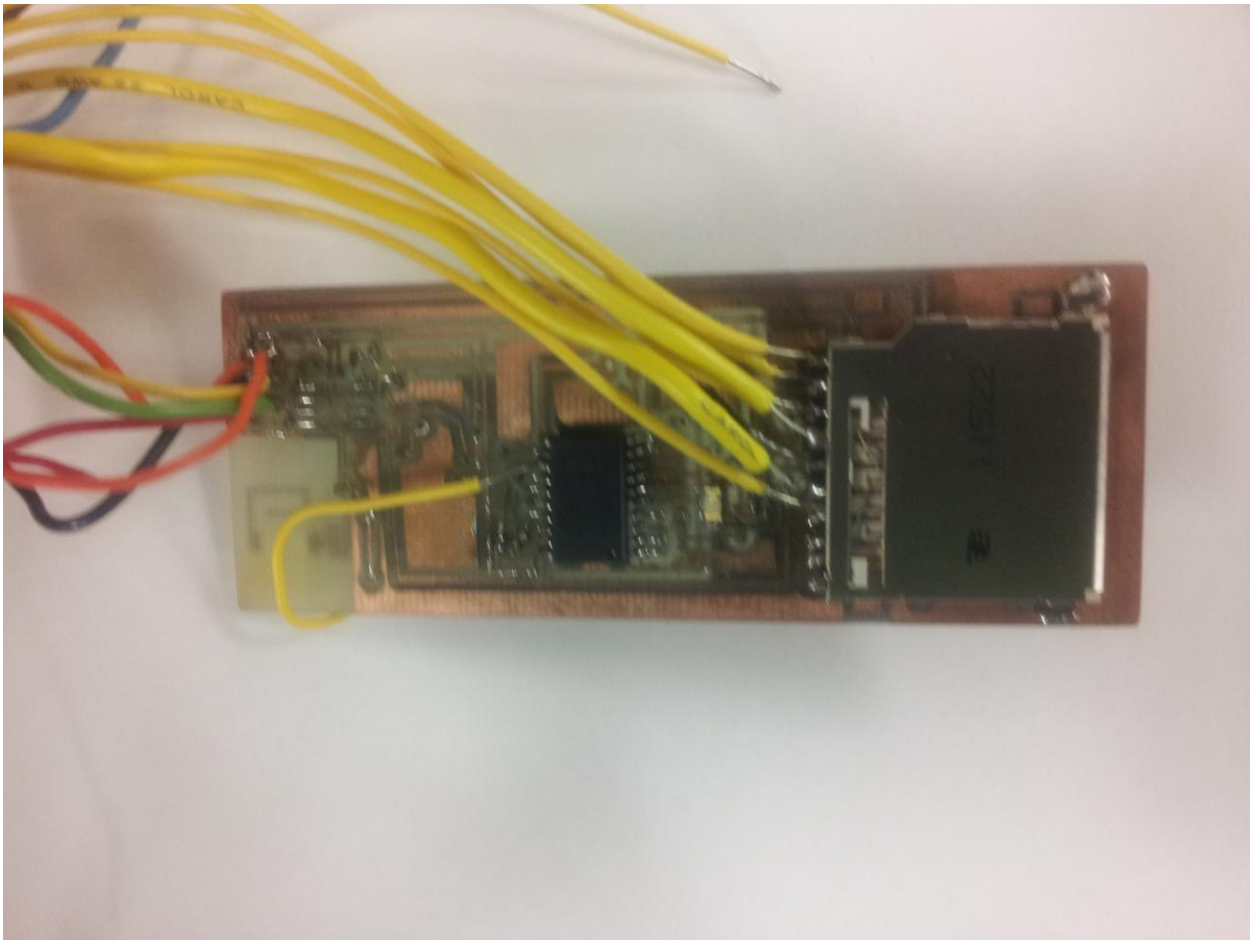


Figure 22: Base Station with test connections