

Behavioral Otter Tracking

By

Jared Lesicko

Kenji Nanto

Miceal Rooney

Final Report for ECE 445, Senior Design, Spring 2012

TA: Mustafa Mir

2 May 2012

Project No. 16

Abstract

When studying wildlife, it is difficult for biologists to track when and where an animal leaves or enters a specific area. This system will identify individual otter movement across a predetermined boundary, while recording time and ambient temperature. It will include passive RFID tags on the otter, an antenna-receiver, a microcontroller and a temperature sensor, all powered by changeable batteries. It will then store the data for later reading by the user. This project will cost much less commercial RFID systems and be tailored to user needs.

Contents

1. Introduction.....	4
1.1 RFID Tag/Receiver	4
1.2 Power	5
1.3 Control and Data Storage.....	5
2. Design	5
2.1 Block Descriptions.....	5
2.1.1 RFID Tag/Receiver	5
2.1.2 12 Volt Power Supply and Voltage Regulator.....	8
2.1.3 9V Battery	9
2.1.4 Control Unit	9
2.1.5 Temperature Sensor.....	10
3. Verification.....	10
3.1 Receiver	10
3.2 12V Power Supply and Voltage Regulator.....	11
3.3 9V Battery	12
3.4 Control Unit	12
3.5 Temperature Sensor.....	14
4. Costs.....	15
Cost Analysis:	15
5. Conclusion	16
5.1 Accomplishments and Challenges	16
5.2 Ethical Considerations	16
5.3 Future Work	16
References.....	17
Appendix A Requirements and Verification Table.....	18
Appendix B Schematics and Tables.....	20
Appendix C Interfacing With the RFM/CTL.....	23
Installation and Setup.....	23
Operation.....	24
Appendix D Final Coding for Arduino Mega 2560	26

1. Introduction

The motivation of this project is to give The Prairie Research Institute a system that can determine the time and ambient temperature when individual otters get in and out of a local pond. Very little behavioral research has been done on the North American River Otter in Illinois, and many commercially available tracking systems are extremely expensive. This system will provide a low cost alternative.

Due to their physiological aspects and environment, otters present an interesting challenge to monitor their behavior. Otters' heads are smaller than their necks; therefore, the most popular tracking device, a collar, is impractical. In addition, they spend significant time both in water and on land. This causes significant attenuation to a signal generated by a transmitter. In addition, otters will chew off devices attached to their paws or tails.

After extensively looking into the topic, we have found no other researchers who utilize RFID tags with otters. This project also has a unique goal in tracking specific behavior, rather than tracking real time location, such as by using GPS, which is also very expensive.

The goal of this project is to sense and record the time and ambient temperature when an otter enters or exits the pond. An antenna will sense the tag inside the otter and communicate the tag ID to a central control unit. When the control unit receives this data, it will record the ID number, temperature, and time. The user can then extract the data via an SD card.

Benefits:

- Can track otter movement without requiring user presence
- Approximately one fifth the price of a commercial RFID reader system
- Assists biologists' understanding of otter behavior
- Temperature dependent behavior tracking

Features:

- Recognition of RFID tag in outdoor environment
- Simultaneously records tag ID, time, and ambient temperature.
- Provides adequate onboard storage for tracking multiple individuals for 3 days
- Low power design for continuous use for up to 3 days
- Provides easy access through SD card

There were two different design routes used in this project. Each design will be mentioned, yet only the one, which had far more success, will be concentrated on more.

1.1 RFID Tag/Receiver

The project is composed of two different receiver designs. The first design, which failed to work, consists of single chip Read/Write base station, external circuitry, self-built antenna and a FDX-B PIT (Passive Integrated Transponder) tag. The second receiver is commercially bought and includes an antenna, driver/RFM (Radio Frequency Module), a control unit and HDX TIRIS PIT tag.

1.2 Power

The main power supplies are a 12V car battery and 2 9V batteries. The 12V car battery includes a voltage regulator.

1.3 Control and Data Storage

The main processing unit of the project is an Arduino Mega 2560. Attached to the top of the Arduino Mega is the Adafruit Data Logging Shield. The Data Logging Shield has a SD card port for easy data access.

2. Design

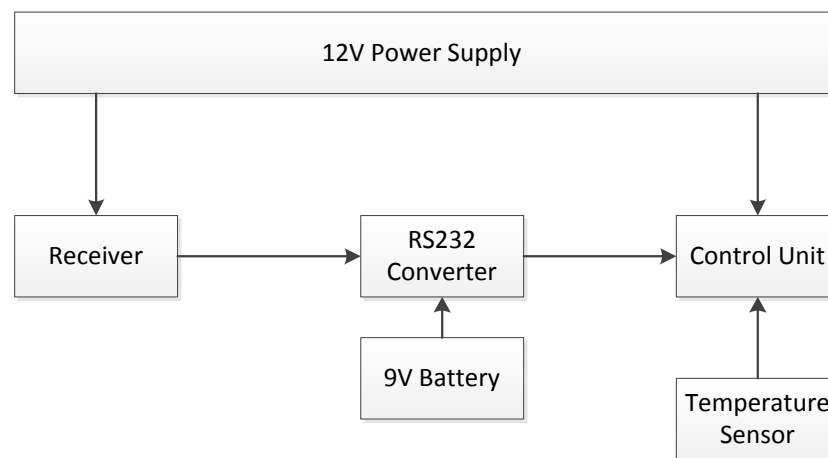


Figure 1: Block Diagram

2.1 Block Descriptions

2.1.1 RFID Tag/Receiver

After the commercial receiver was bought, we changed from the BIOMARK HPT22 Passive Integrated Transponder (PIT) to a TIRIS RI-TRP-DR2B made by Texas Instruments. Physically, the TIRIS transponder works the same as the BIOMARK transponder. Both are sensed by inductive coupling at 134.2 KHz, both are encapsulated by glass and both are Read Only. The main difference is the TIRIS transponder has 64 read only bits and utilizes the HDX protocol instead of the FDX-B. HDX protocol uses FSK (Frequency Shift Key) modulation. The most beneficial aspect of using a commercial receiver is the receiver demodulates and decodes. Therefore, the demodulating/decoding process never had to be researched.

The first receiver included the antenna to sense the RFID tag, an Atmel U2270B Read/Write Base Station, and complementing circuitry. The antenna is a resonant loop designed to operate at 134.2 kHz. Because of the extremely long wavelength, it utilizes inductive coupling to sense a load (PIT tag). When the tag passes over, the antenna's signal is ASK modulated. The receiver

circuit then demodulates the signal and routes it to the U2270B. The base station then digitizes the signal and sends the encoded data to the microcontroller. Using a capacitance of 1nF and a frequency of 134.2 kHz, the required inductance, L , can be calculated:

$$L = \frac{1}{(2\pi f)^2 C} = \frac{1}{(2\pi \times 134.2kHz)^2 \times 1nF} = 1.406mH \quad (Eq. 1)$$

The number of turns is then calculated using the inductance and a radius of 8 cm

$$N = \sqrt{\frac{L}{\pi \mu_0 r}} = \sqrt{\frac{1.406mH}{\pi \mu_0 \times 8cm}} = 66.7 \text{ turns} \quad (Eq. 2)$$

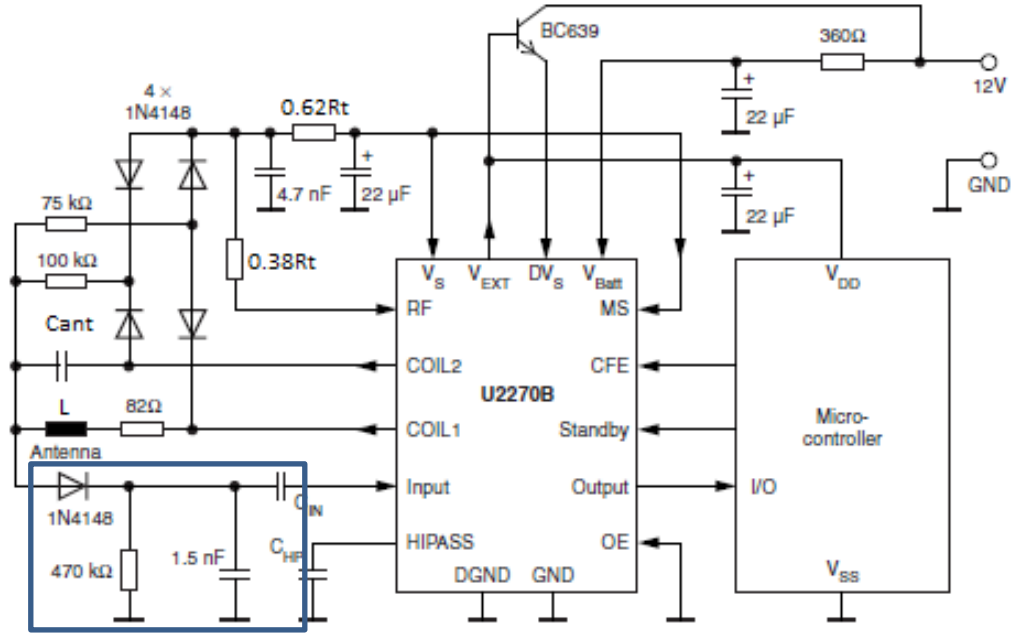


Figure 2: Original Receiver Circuit, envelope detector shown in blue [1]

$$\begin{aligned} L &= 1.406mH \\ C_{ant} &= 1nF \\ C_{IN} &= 680pF \\ C_{HP} &= 100nF \end{aligned}$$

$$R_t[k\Omega] = \frac{14375}{f[kHz]} - 5 = \frac{14375}{134.2kHz} - 5 = 102k\Omega$$

(Eq. 3)

The second receiver design system included a commercially bought 27uH antenna, a RI- RFM-007b (Radio Frequency Module), a RI-CTL-MB2B controller and a MAX232N RS-232 to TTL converter. All previous mentioned products, except for the RS-232-to-TTL converter, were purchased from Texas Instruments. The signal path of this receiver can be traced as follows. The Antenna is driven by the RFM at 134.2 KHz. Once the PIT tag is placed over the antenna, the 64 bit ID is read into RFM, where it is amplified and demodulated. The demodulated signal is then sent to CTL controller, where it is decoded and converted into ASCII characters. These ASCII characters are serially output at a baud rate of 9600 bps from the RS-232 output into the MAX232N, which converts them to TTL and inputs the ASCII data into the controller for processing [2].



Figure 3: 27 uH Antenna

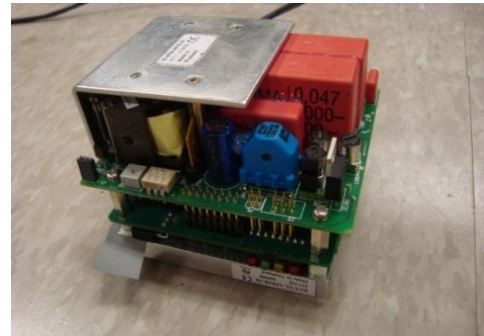


Figure 4: RFM (top) and CTL (bottom)

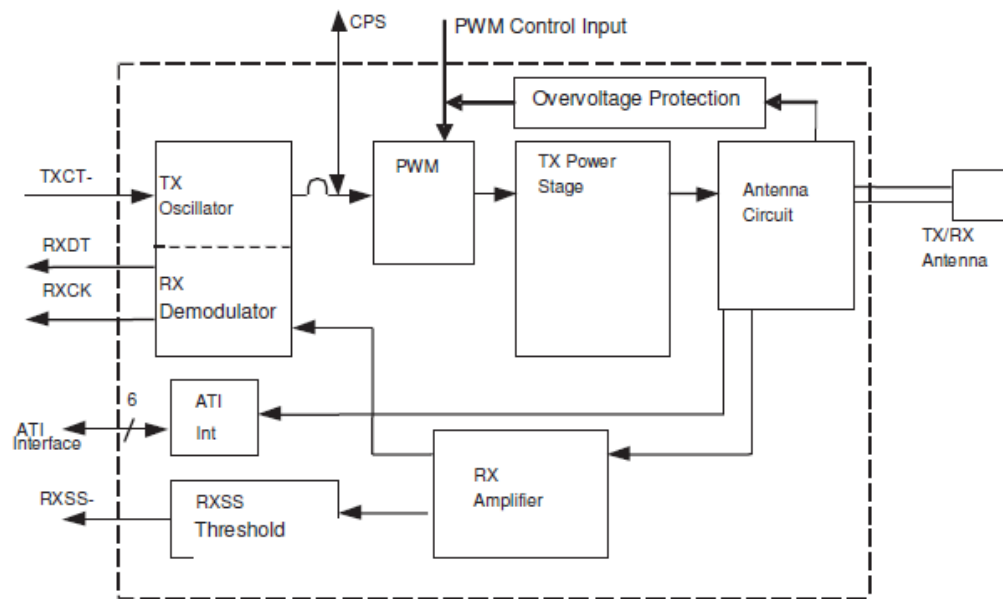


Figure 5: RFM Schematic [3]

2.1.2 12 Volt Power Supply and Voltage Regulator

This power supply will provide more power than the one powering the main controller. It will need to power multiple antennas and withstand a large current draw. The rating needed can be determined by calculating the power consumption of the receiver, microcontroller, and the temperature sensor. The following calculations are based on maximum current draw.

$$\text{Microcontroller Power [W]} = \text{Max Current Draw [mA]} \times 12V = 200\text{mA} \times 12V = 2.4W \quad (\text{Eq. 4})$$

$$\begin{aligned} \text{Temperature Sensor Power [W]} &= \text{Temperature Sensor Current Draw [mA]} \times 3.3V \\ &= 0.05\text{mA} \times 3.3V = 1.65 \times 10^{-4}W \end{aligned} \quad (\text{Eq. 5})$$

$$\text{RFM and CTL Power [W]} = \text{Max Current Draw [mA]} \times 12V = 300\text{mA} \times 12V = 3.6W \quad (\text{Eq. 6})$$

$$\begin{aligned} \text{12V Battery Rating [Ah]} &= \frac{\text{Total Power Used [W]} \times 72 \text{ hrs}}{12V} \\ &= \frac{(\text{Microcontroller} + \text{Temp Sensor} + \text{RFM and CTL}) \times 72\text{hrs}}{12V} \\ &= \frac{(2.4W + 1.65 \times 10^{-4}W + 3.6W) \times 72\text{hrs}}{12V} = 36Ah \end{aligned} \quad (\text{Eq. 7})$$

In order to satisfy this requirement, a car battery is used. The Prairie Research Institute has provided Everstart Marine batteries rated at 75 Ah. During regular use, the system should not normally use the maximum values noted; therefore, this battery will last for far longer than the required three days.

The voltage regulator is necessary for the RFM-007B. The RFM requires the RMS voltage to be less than 50mV. Additionally, the regulator must be able to handle large amounts of current (0.5A) while providing low dropout in order to maintain the 12V supply. For these reasons, the Fairchild Semiconductor KA378R12CTU is used. The regulator can supply 3A, and at max current draw it drops 0.5V [4].

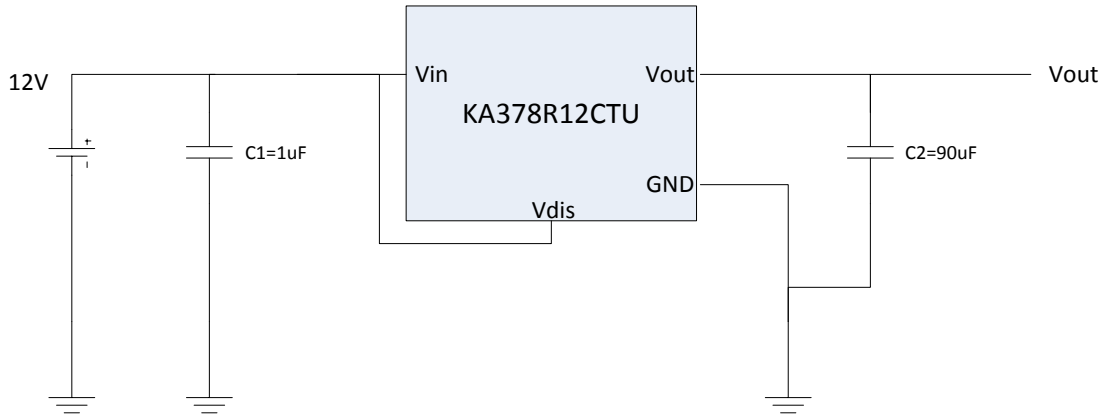


Figure 6 Voltage Regulator Circuit

C1 is only necessary if the regulator is far from the source, and C2 improves stability [1]. V_{dis} disables the device when low; therefore, it is connected to V_{in} (there is no need to disable the regulator).

2.1.3 9V Battery

The Max-232 chip requires two voltage sources to implement the logic level change. There is an allowed range of 4.7V to 15V for the positive supply and -0.3V to -15V for the negative supply [2]. We use standard 9V batteries due to their cost and availability.

$$9V \text{ Battery Rating [mAh]} = 1mA \times 72hrs = 72mAh \quad (Eq. 8)$$

The Energizer Industrial battery has a 600mAh rating at 25mA, so the batteries should last for more than a month [5].

2.1.4 Control Unit

The control unit consists of the microcontroller, the data logger, and the SD card. The microcontroller is an Arduino Mega 2560 board based on the ATmega2560. It has 56 digital I/O, 16 analog inputs, 4 serial ports, a 16 MHz oscillator, and operates at 12 VDC which is supplied from the car battery. This is the central control unit that processes the data from the receiver, the temperature sensor, and the real time clock on the data logger. When the receiver senses a tag, the microcontroller will decode, process, and send the data to the data logger for recording.

When starting with the first design, the control unit was expected to be able to demodulate the ASK signal but with the new receiver unit, this is not needed. With the Texas Instrument receivers, the control module will only receive bit ASCII letters. In the application of this project, the receiver uses the line command which will consistently send an "L" or an "LI", telling the control module that it is reading while sending these letters. When a tag is sensed the receiver

will send a “LR” to the control unit followed by 22 ASCII numbers, giving us the ID number. When waiting for the “LR,” the control unit will just wait for incoming “R,” since that letter is never used for any other command and will not be used in an ID number. Once the “R” is sensed, the control module then takes the voltage reading of the temperature sensor, calculates the Fahrenheit temperature, reads the time stamp on the DS1307 real-time chip, mounted on the Adafruit data logger shield and then stores it into a .csv. The .csv file is a file easily used in Microsoft Excel and can help the user manipulate the data being stored. Once this data is stored the control unit waits for the next incoming “R.”

The code in Appendix D requires specific libraries. These two libraries help the software serial access work with the DS1307. Most of the code that is used to read the time stamp was taken from the example “SoftDS1307” on the Adafruit forums. [6]

2.1.5 Temperature Sensor

The temperature sensor is an Analog Devices TMP36 Temperature Sensor. At -50°C, it outputs 0V, and at 125 °C it outputs 1.75V. This signal will be processed by the microcontroller and recorded by the data logger when needed. To determine the temperature in Celsius, the following equation is used [7]:

$$Temperature [^{\circ}C] = \frac{V_{out}[mV] - 500mV}{10mV/^{\circ}C} \quad (Eq.9)$$

The V_{out} can be determined by using the 10 bit value of the Analog to Digital Converter (ADC) on the microcontroller and the supply voltage:

$$Voltage\ at\ Pin\ [mV] = \frac{ADC\ Reading}{1024} \times 3300mV \quad (Eq.10)$$

3. Verification

3.1 Receiver

In order to verify that the receiver is properly working, one must confirm that the antenna inductance is properly tuned, the RS-232 port has the proper output voltage levels and ASCII waveforms and the MAX232N chip properly converts from RS-232 to TTL voltage levels.

When bought, the antenna inductance is advertised as 27uH. The receiver will properly detect a tag if the inductance of the antenna is within-26-29uH. When we placed the antenna in the LRC meter, the inductance readout was 26.6 uH.

If the receiver’s RS-232 port is working properly, while in line mode, one should observe a waveform with a peak-to-peak voltage greater than 6V but less than 25V with a ASCII ‘L’ bit pattern of “01001100” inverted. Before any voltage waveforms can be extrapolated, the CTL

module must be placed in line mode. The step-by-step process of line mode can be seen in Appendix C. Once the CTL module is in line mode, an oscilloscope was used to probe pins 3(GND) and 4(RS-232 Data Output) on the ST21 port on the CTL module. The waveform seen below verifies an ASCII output L.

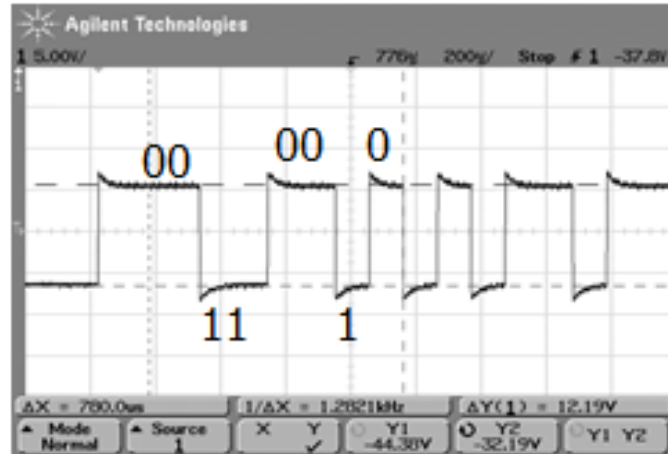


Figure 7: RS-232 output of ASCII character "L"

Next, pins 12(TTL Output) and 15(GND) of the MAX232N were probed by an oscilloscope. The waveform below shows a TTL ASCII 'L' waveform.

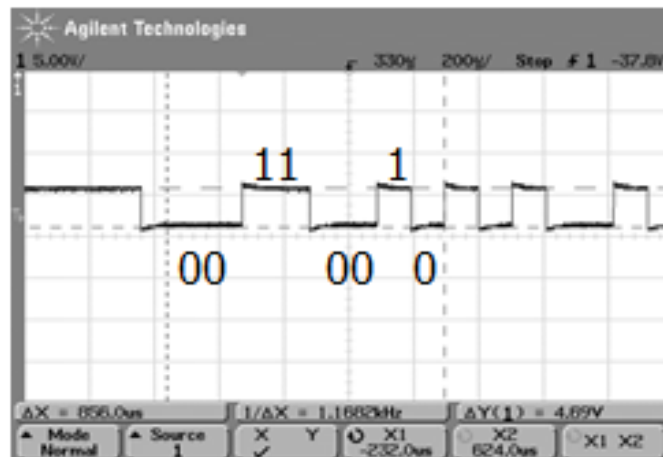


Figure 8: TTL signal after conversion from RS-232

3.2 12V Power Supply and Voltage Regulator

The 12 volt power supply must last for 72 hours and output a ripple voltage of less than 50mV. The battery and regulator were connected to a DMM, providing a reading of $V_{rms} = 0.5mV$. This is well below the required 50mV.

The current draw was determined by connecting the receiver to a power supply, and measuring the voltage across a 1Ω resistor placed between the power supply and the system. As a result of Ohm's Law, $V = IR$, the voltage across this resistor is equal to the current drawn by the circuit.

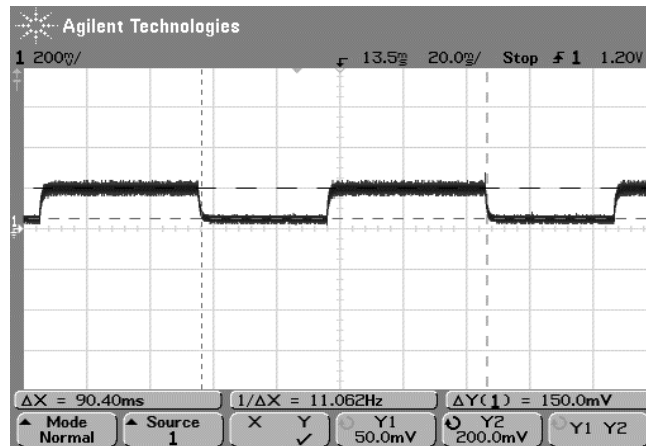


Figure 9: Receiver Current Draw

As seen in the Figure 9, the maximum current draw by the receiver is 200mA (corresponding to $Y2 = 200mV$). This is well below the estimated maximum current draw in the Equation 5. The maximum allowed current draw from the Arduino is 200mA [8].

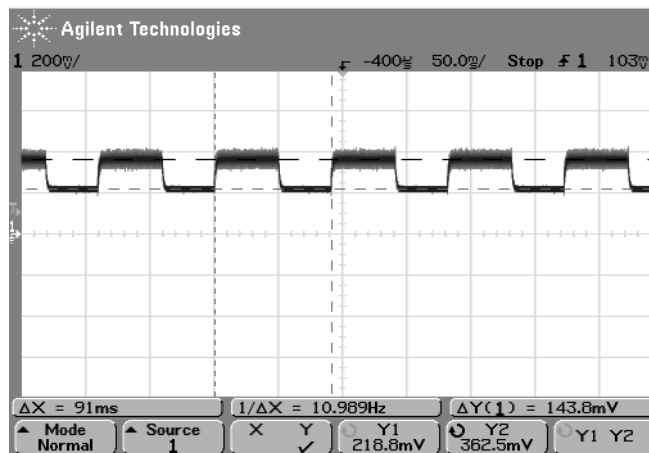


Figure 10: Total system current draw

The figure above shows that the total current draw is 362.5mA (at a maximum), validating the lifespan of the battery.

3.3 9V Battery

The 9V battery must also last for 72 hours. As mentioned in section 2.1.3, the estimated current draw is 1mA. When the terminals of the circuit are connected to a DC source, the measured output is also 1mA, providing a requirement of 72mAh, well below the battery rating of 600mAh (at 25 mA draw).

3.4 Control Unit

One thing the control unit must be able to do is store at least 150 entries of tag reads. This will be done by making sure that the SD memory card used for memory storage has to be at least 100 MB. This was verified by purchasing a 2 GB SD card. Another important verification is to

show that the control unit can correctly identify the RFID. This was demonstrated by running the receiver and connecting the RS232 output of the receiver and converting it to the MAX232 converter to change the serial data into TTL logic. Once this was obtained we then connected that output to the control unit and test the incoming data with the following code.

```
void setup() {  
  // initialize both serial ports:  
  Serial.begin(9600);  
  Serial1.begin(9600);  
}  
  
void loop() {  
  // read from port 1, send to port 0:  
  if (Serial1.available()) {  
    int inByte = Serial1.read();  
    Serial.write(inByte);  
  }  
}
```

When running this code with the microcontroller, ASCII characters were read from the receiver and see it through the serial monitor on the computer connected to the control unit and the following is what was received.

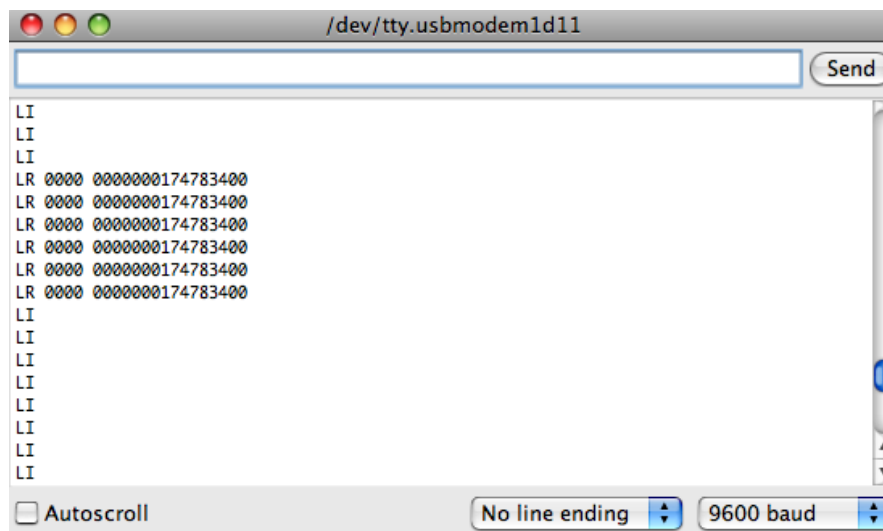


Figure 11: Serial Monitor for Control Unit connected to Receiver

When trying to manipulate the data being read to store the Tag's ID number and then store the Temperature and Time stamp, the final problems are run into. Storage is successful when a tag is read and input the time and temperature, but the ID data is changed. This can be seen in the following figure.

4. Costs

Cost Analysis:

Table 1: Cost of Labor

Name	Hourly Rate	Total Hours	Total = Hourly Rate x 2.5 x Total Hours
Jared Lesicko	\$40	200	\$20000
Kenji Nanto	\$40	200	\$20000
Miceal Rooney	\$40	200	\$20000
Total			\$60000

Table 2: Bill of Materials

Part Number	Description	Cost	Quantity	Total
Arduino Mega 2560	Microcontroller Board	\$58	1	\$58
Adafruit Data Logger	Data Logger	\$20	1	\$20
Analog Devices TMP36GT9	Temperature Sensor	\$2	1	\$2
Patriot PSF2G40SD	2 GB SD Card	\$5	2	\$10
RFM-007B	Driver/Receiver	\$350	1	\$350
CTL-MB2B	Controller	\$320	1	\$320
-	Capacitors	\$1	7	\$7
-	Casing	\$100	1	\$50
	Wires	\$2	1	\$2
UPG 75aH	Battery	\$170	1	\$170
ANT-G01E-30	Antenna	\$240	1	\$240
TRP-RR2B-30	Transponders	\$5.46	5	\$27.20
Fairchild Semiconductor KA378R12CTU	Voltage Regulator	\$.57	1	\$.57
MAX232N	RS-232 to TTL	\$.62	1	\$.62
Radioshack Circuit Board	Circuit Board	\$4	1	\$4
Total				\$1,261

Table 3: Total Costs

Section	Total
Labor	\$60000
Parts	\$1261
Grand Total	\$61261

5. Conclusion

5.1 Accomplishments and Challenges

The Otter Sensing Project is an almost total success. The system can sense an RFID tag 100 percent of the time, with a range of approximately nine inches, allowing it to be buried. The coverage of the antenna may only be three feet out of the approximately 70 foot shoreline, but otters generally use the same paths, specifically around their latrine area. In addition the system can last for much longer than the required duration. Even at max power usage, it will still last for six days during the summer.

Many problems were encountered during the design of the system. The initial implementation included designing the antenna and the receiver. Due to the difficulties in obtaining correct signal output and antenna resonance, this setup was scrapped in favor of a commercial receiver and antenna. While the choice increased the cost, it resulted in a much more reliable system. Currently, the project does not have a final casing to be mounted in. Lastly, recording the tag ID at the same time as recording the time and temperature has not been implemented effectively; the data is corrupted when stored.

5.2 Ethical Considerations

This project does not encounter very many ethical problems. The main concern is to avoid harming or interfering with the otters' lives. All electrical equipment must be enclosed in order to ensure the animals do not shock or burn themselves. Otters are not frightened by human activity or man-made objects; therefore, the antenna and main circuitry will not interfere. Additionally, this project may be used without requiring any licenses; the FCC does not require one for low frequency RFID products.

5.3 Future Work

The immediate goal is to obtain a field able project for the Prairie Research Institute. In order to accomplish this, the waterproof casing must be obtained, and the code must be corrected. Beyond the completion of this project, additional antennas may be used along with a multiplexer available from Texas Instruments to increase the coverage. A future project could involve synchronizing these extra antennas and handling the much larger power requirements. The system can also be duplicated for any other researchers interested in its capabilities.

References

- [1] *Read/Write Base Station U2270B*, datasheet, Atmel Corporation, 2008. Available at: <http://www.atmel.com/Images/doc4684.pdf>
- [2] *MAX232 DUAL EIA-232 DRIVERS/RECEIVERS*, datasheet, Texas Instruments, 2004. Available at: <http://www.ti.com/lit/ds/symlink/max232.pdf>
- [3] *Series 2000 Reader System High Performance Reader Frequency Module RI-RFM-007B*, reference guide, Texas Instruments, 2006. Available at: <http://www.ti.com/lit/ug/scbu022/scbu022.pdf>
- [4] *KA378R12C Low Dropout Voltage Regulator*, datasheet, Fairchild Semiconductor Corporation, 2005. Available at: <http://www.fairchildsemi.com/ds/KA/KA378R12CTU.pdf>
- [5] *ENERGIZER EN22*, datasheet, Energizer Holdings, Inc. Available at: <http://data.energizer.com/PDFs/EN22.pdf>
- [6] New i2c libraries with 'softi2c', web page. Available at: <http://forums.adafruit.com/viewtopic.php?f=25&t=13722>. Accessed April 2012.
- [7] *Low Voltage Temperature Sensors TMP36*, datasheet, Analog Devices Inc., 2010. Available at: http://www.analog.com/static/imported-files/data_sheets/TMP35_36_37.pdf
- [8] *8-bit Microcontroller ATmega2560/V*, datasheet, Atmel Corporation, 2011. Available at: <http://www.atmel.com/Images/doc2549.pdf>
- [9] *Series 2000 Reader System Control Modules RI-CTL-MB2B*, datasheet, Texas Instruments, 2008. Available at: <http://www.ti.com/lit/ug/scbu044/scbu044.pdf>
- [10] *Series 2000 Read System ASCII Protocol*, Reference Guide, Texas Instruments, 2006. Available at: <http://www.ti.com/lit/ug/scbu028/scbu028.pdf>

Appendix A Requirements and Verification Table

Table 4: Requirements and Verification Table

Requirement	Verification	Verification Status (Y or N)
1. System is powered by 12V car battery correctly. a. Battery must have capacity for 3 days of continuous use b. Ripple voltage must be less than 50 <u>mVrms</u>	1. Car battery must last long enough for 3 days of continuous use by system and must be properly regulated. a. Compare battery Ah rating to required Ah. Battery rating is 75Ah, system requirements are below. b. Confirm ripple voltage from regulator output is less than 50 <u>mVrms</u> by connecting output to DMM.	1. Y a. Y b. Y
2. Receiver and controller outputs the correct RS-232 ASCII data signal from RFID tag. a. Antenna has proper inductance of 27uH. b. Insure proper RS-232 voltage levels are being transmitted. c. Identify the proper ASCII code for L.	2. The RFM (Radio Frequency Module) or receiver/antenna driver that is being used accepts an antenna of 26-28uH to insure that proper resonant frequency is being achieved. RS-232 can be anywhere from ± 25 Volts. This is inverted from a normal TTL signal. The TTL ASCII code for L is (1001100)(0), which a. The antenna terminals can be hooked up to an LRC meter to verify that the commercial antenna gives the proper inductance. b. Pin 3(GND) and Pin 4(RS-232 Output) can be hooked to an oscilloscope to observe the <u>Vpp</u> of the waveform. c. Capture one ASCII pulse on the oscilloscope and decode by hand to see if an L is properly transmitting.	2. Y a. Y b. Y c. Y
3. Confirm that the RS-232 ASCII waveform is properly converted to TTL by MAX232N chip. a. <u>Vpp</u> of TTL output from the MAX232N chip 5 <u>Vpp</u> .	3. The MAX232N chip has <u>two RS-232 to TTL inputs</u> . a. Connect Pin 12 (TTL output) and GND to oscilloscope and observe that the proper ASCII waveform (mentioned above) is now TTL. Logic lows must be under 0.8V, and logic highs must be between 2.2V and 5V.	3. Y a. Y

<p>4. Temperature Sensor senses correct temperature within 3°C.</p> <p>a. Sensor draws 0.05 mA from 3.3V supply</p>	<p>4. Connect sensor to 3.3V supply and DMM. Confirm output voltage to $\pm 30mV$ by observing voltage, performing voltage-Celsius conversion, and comparing to thermostat room temperature.</p> <p>a. Connect sensor to 3.3V source and hook up to DMM. Confirm current draw at $0.05 \pm 0.005mA$</p>	<p>4. Y</p> <p>a. Y</p>
<p>5. Control unit records correct tag ID, time, and temperature onto SD card.</p> <p>a. SD card capacity is sufficient to hold the required amount of data (150 entries)</p> <p>b. Correctly identify RFID</p> <p>c. Store all data to SD card in CSV file</p> <p>i. Displays Temperature in SD card on command</p> <p>ii. Displays Time in SD card on command</p> <p>iii. Displays RFID tag in SD card on command</p>	<p>5. Pass tag over receiver 10 separate times under different temperatures and manually record time, temperature, and tag ID. Remove SD card from slot and open the loaded file in Microsoft Excel. Confirm 10 entries in file with correct values.</p> <p>a. Using arbitrary inputs in <u>program</u>, create 150 entries to log onto the SD card. Remove SD card and read the file with Excel. Confirm 150 entries.</p> <p>b. Assuming data is successfully <u>decoded</u>, use the serial ASCII monitor of the <u>Arduino</u> software program to observe output.</p> <p>c. When CSV file is opened, all required data is present</p> <p>i. When commanded to, check SD memory card to see temperature recorded.</p> <p>ii. When commanded to, check SD memory card to see time recorded.</p> <p>iii. When commanded to, check SD memory card to see RFID tag recorded.</p>	<p>5. N</p> <p>a. Y</p> <p>b. Y</p> <p>c. N</p> <p>i. Y</p> <p>ii. Y</p> <p>iii. N</p>

Appendix B Schematics and Tables

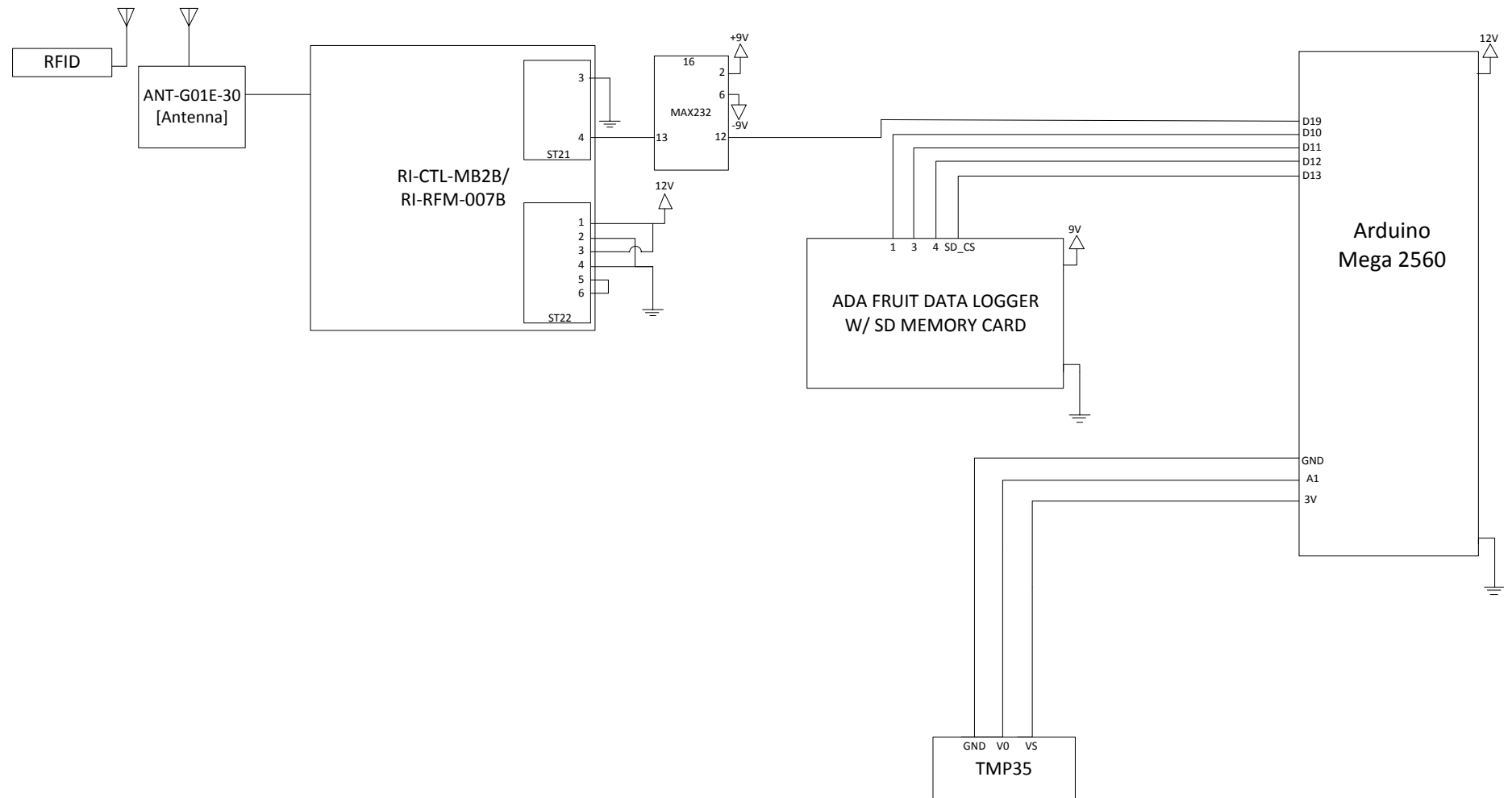


Figure 14: System Pinout Diagram

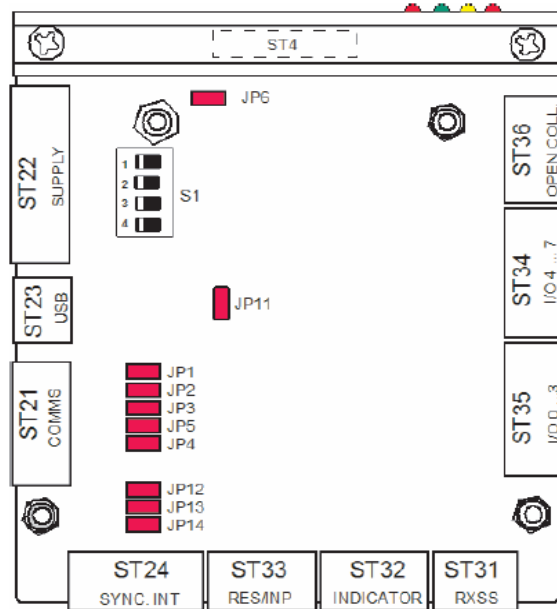


Figure 15: Bottom View of CTL [9]

Table 5: ST21-RS-232 Communication Interface [9]

Signal Name	Pin	Comment
RXD	1	RS-232-C Serial Data Input
DTR	2	RS-232-C Data Terminal Ready Input
GND	3	Signal Ground
TXD	4	RS-232-C Serial Data Output
DSR	5	RS-232-C Data Set Ready Output

Table 6: ST22-Supply [9]

Signal Name	Pin	Comment
VSP	1	Supply voltage for the RF Module
GNDP	2	Ground line for the RF Module supply.
VDC	3	Non-regulated supply voltage for the control logic circuitry.
GND	4	Signal ground line for the control logic supply.
VCC2	5	Regulated supply voltage (5VDC) for the control logic circuitry.
VCC3	6	Memory data retention supply voltage
GND	7	Signal ground

Table 7: ST32-Indicator Outputs [9]

Signal Name	Pin	Comment
VCC2	1	Regulated 5 VDC supply output
ACTIVE-	2	Open Collector Output: RF Module Transmitter Signal
OK-	3	Open Collector Output: OK Signal
EMI-	4	Open Collector Output: EMI Signal

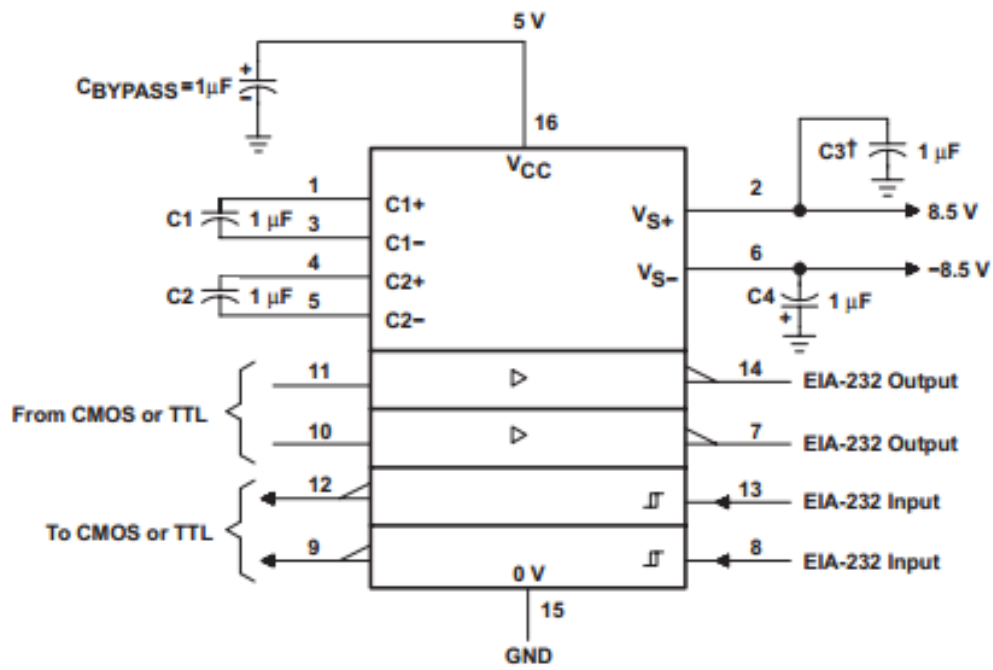


Figure 16: MAX232N Circuit [2]

Appendix C Interfacing With the RFM/CTL

Installation and Setup

1. Connect the RFM to the computer using the USB-Micro USB cable. On first time setup, the RFM drivers should install automatically.
2. Download the file **putty.exe** located under “For Windows on Intel x86” from the PuTTY website: <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>.
3. Run **putty.exe**. The following screen will appear:

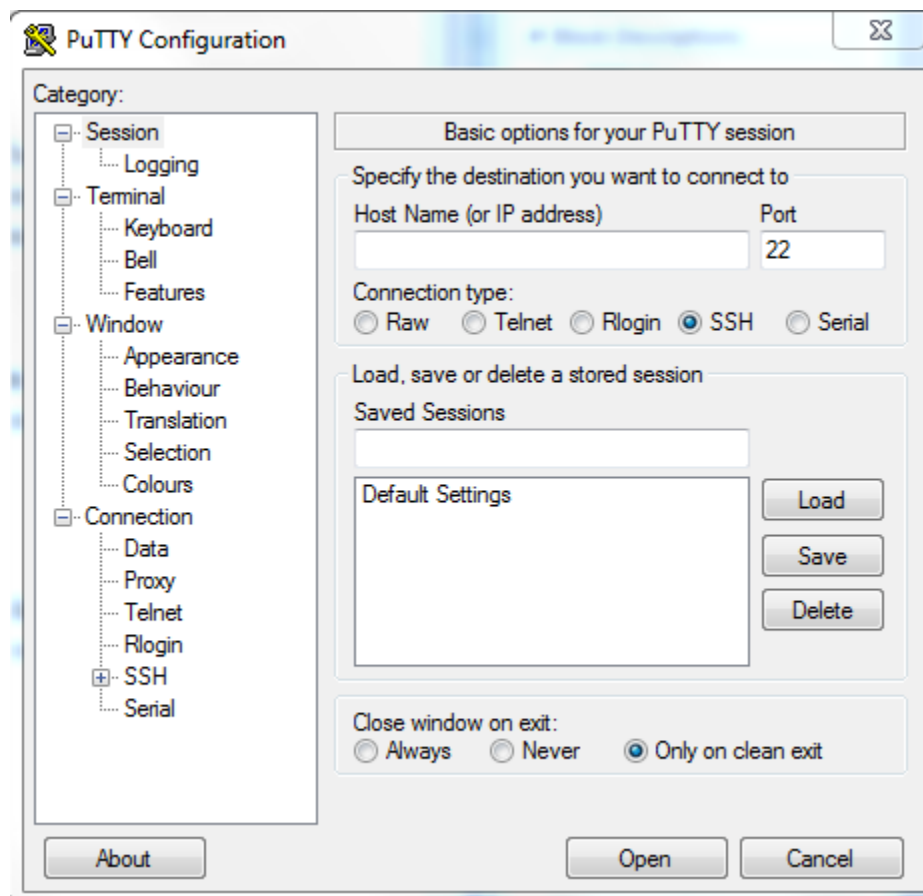


Figure 17: Initial PuTTY Configuration Screen

4. The receiver communicates with the computer by way of serial communication. Change “Connection type” to Serial. Then under “Serial line” type “COM4”. Click Open.

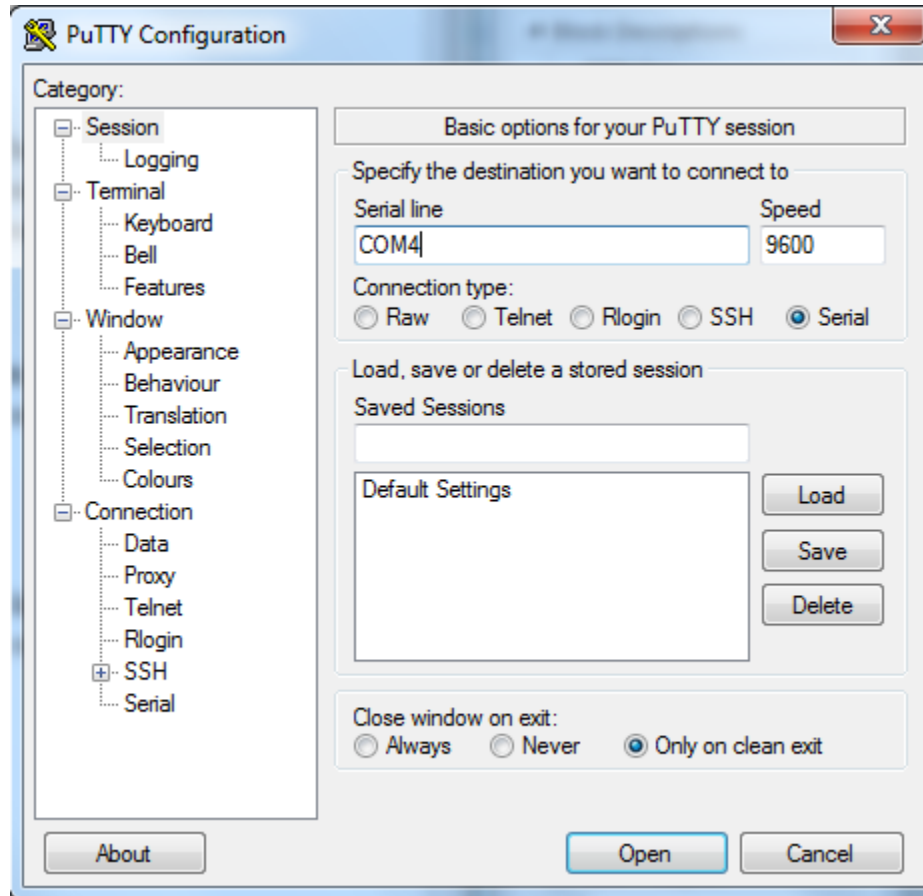


Figure 18: Final PuTTY Configuration Screen

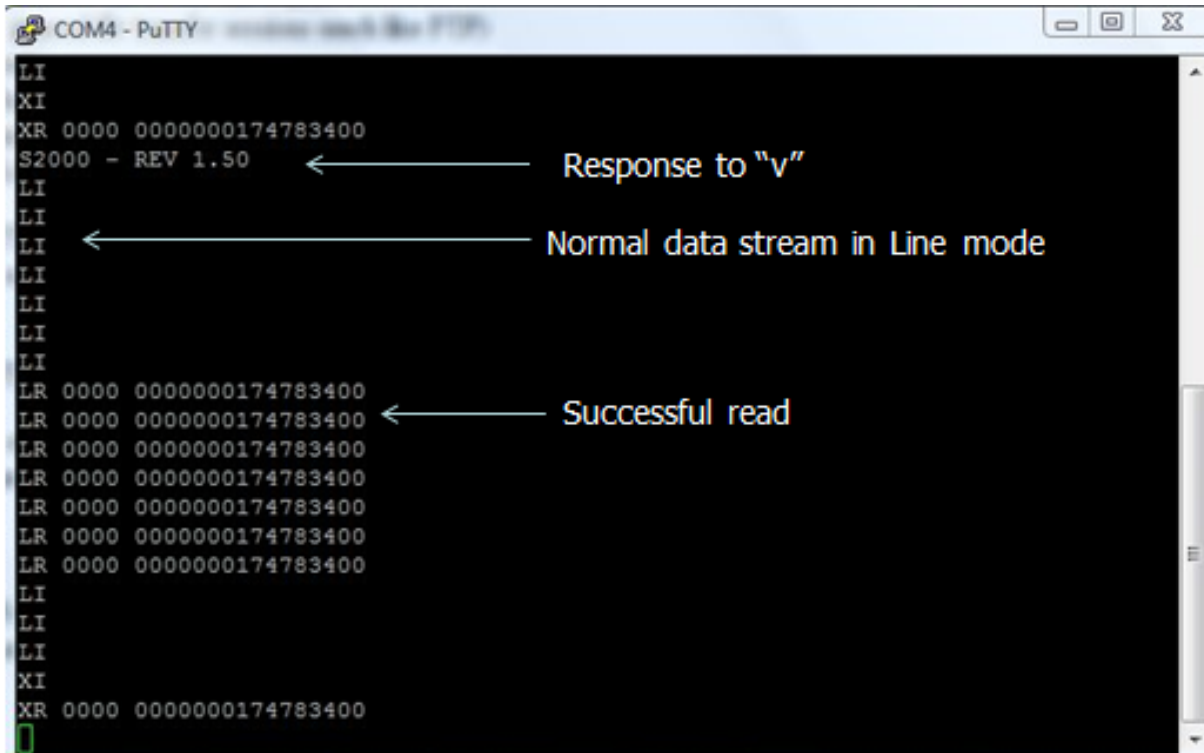
Operation

The terminal accepts commands given by the single press of a key, using ASCII communication protocol. For this system’s purposes, only V, X, and L will be used. For a full list of commands, see the ASCII Reference Guide [10]. When giving a command, the terminal does not show the key pressed, but only the response from the receiver.

- V: Requests the version number of the RFM. The receiver should respond with something similar to “S2000 – REV 1.50”. This command is useful to confirm that the receiver is connected to the computer.
- X: This puts the receiver in Execute mode. It attempts a single read then goes to idle. It is useful to stop the receiver. The following are possible responses:
- X –Nothing is read.
 - XI –Some signal is read but not a tag, generally due to electromagnetic noise.
 - XR 0000 0000000174783400 –A successful tag read with the tag’s ID.

- L: This command puts the receiver in Line mode. Line mode is the normal mode of operation for this system. The RFM will continually attempt to read a signal. When a signal is read it will immediately send it. Possible responses are:
- L –Nothing is read.
 - LI - Some signal is read but not a tag, generally due to electromagnetic noise.
 - LR 0000 00000000174783400 –A successful tag read with the tag’s ID.

Below is an example of the terminal screen.



```
COM4 - PuTTY
LI
XI
XR 0000 00000000174783400
S2000 - REV 1.50
LI
LI
LI
LI
LI
LI
LI
LR 0000 00000000174783400
LR 0000 00000000174783400
LR 0000 00000000174783400
LR 0000 00000000174783400
LR 0000 00000000174783400
LR 0000 00000000174783400
LR 0000 00000000174783400
LI
LI
LI
XI
XR 0000 00000000174783400
```

Figure 19: Sample terminal screen

Once the system is confirmed working, the user can disconnect the computer from the receiver. The system is now ready to be used.

Appendix D Final Coding for Arduino Mega 2560

```
#include <SD.h>
#define aref_voltage 3.3
const int chipSelect = 4;
int tempPin = 0;
int n=0;
int tempReading;
int check;
char Buf='X';
char CurrentID[]="0000 0000000000000000";
char ID[]="174783400";
char incomingByte;

//-----
//-----
/**SoftDS1307_example_Headers
//-----
//-----
// Utility sketch to explore DS1307 and
// demonstrate SoftI2cMaster and TwiMaster
//
#include <avr/pgmspace.h>
#include <I2cMaster.h>

// select software or hardware i2c
#define USE_SOFT_I2C 1

#if USE_SOFT_I2C

#if defined(__AVR_ATmega1280__)\
|| defined(__AVR_ATmega2560__)
// Mega analog pins 4 and 5
// pins for DS1307 with software i2c on Mega
#define SDA_PIN 58
#define SCL_PIN 59

#elif defined(__AVR_ATmega168__)\
|| defined(__AVR_ATmega168P__)\
|| defined(__AVR_ATmega328P__)
// 168 and 328 Arduinos analog pin 4 and 5
#define SDA_PIN 18
#define SCL_PIN 19

#else // CPU type
#error unknown CPU
```

```

#endif // CPU type

// An instance of class for software master
SoftI2cMaster rtc(SDA_PIN, SCL_PIN);

#else // USE_SOFT_I2C

// Pins for DS1307 with hardware i2c
// connect SCL to Arduino 168/328 analog pin 5
// connect SDA to Arduino 168/328 analog pin 4

// Instance of class for hardware master with pullups enabled
TwiMaster rtc(true);

#endif // USE_SOFT_I2C

// i2c 8-bit address for DS1307. low bit is read/write
#define DS1307ADDR 0XD0
//-----
/*
 * Read 'count' bytes from the DS1307 starting at 'address'
 */
uint8_t readDS1307(uint8_t address, uint8_t *buf, uint8_t count) {
    // issue a start condition, send device address and write direction bit
    if (!rtc.start(DS1307ADDR | I2C_WRITE)) return false;

    // send the DS1307 address
    if (!rtc.write(address)) return false;

    // issue a repeated start condition, send device address and read direction bit
    if (!rtc.restart(DS1307ADDR | I2C_READ)) return false;

    // read data from the DS1307
    for (uint8_t i = 0; i < count; i++) {

        // send Ack until last byte then send Ack
        buf[i] = rtc.read(i == (count-1));
    }

    // issue a stop condition
    rtc.stop();
    return true;
}

//-----
/** Store and print a string in flash memory.*/

```

```

#define PgmPrint(x) SerialPrint_P(PSTR(x))
/** Store and print a string in flash memory followed by a CR/LF.*/
#define PgmPrintln(x) SerialPrintln_P(PSTR(x))
//-----
/*
 * Print a string in flash memory to the serial
 */
//static void SerialPrint_P(PGM_P str) {
// for (uint8_t c; (c = pgm_read_byte(str)); str++) Serial.write(c);
//}
//-----
/*
 * Print a string in flash memory followed by a CR/LF.
 */
//static void SerialPrintln_P(PGM_P str) {
// SerialPrint_P(str);
// Serial.println();
//}
//-----
// day of week U.S. convention
char *Ddd[] = {"Bad", "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"};

//-----
void hexPrint(uint8_t v) {
  Serial.print(v >> 4, HEX);
  Serial.print(v & 0XF, HEX);
}
void hexPrintln(uint8_t v) {
  hexPrint(v);
  Serial.println();
}
//-----
/*
 * write 'count' bytes to DS1307 starting at 'address'
 */
uint8_t writeDS1307(uint8_t address, uint8_t *buf, uint8_t count) {
  // issue a start condition, send device address and write direction bit
  if (!rtc.start(DS1307ADDR | I2C_WRITE)) return false;

  // send the DS1307 address
  if (!rtc.write(address)) return false;

  // send data to the DS1307
  for (uint8_t i = 0; i < count; i++) {
    if (!rtc.write(buf[i])) return false;
  }

  // issue a stop condition

```

```

    rtc.stop();
    return true;
}
//-----
//-----
//-----

void setup(void) {
    Serial.begin(9600); //Serial output to Arduino Mega 2560 for data transfer
    Serial1.begin(9600); //Serial input from receiver
    Serial.print("Initializing SD card...");
    // make sure that the default chip select pin is set to
    // output, even if you don't use it:
    pinMode(10, OUTPUT);

    // see if the card is present and can be initialized:
    if (!SD.begin(chipSelect)) {
        Serial.println("Card failed, or not present");
        // don't do anything more:
        return;
    }
    Serial.println("card initialized.");
}

//-----
//-----
//-----

void loop(void) {
    while(check!=1){
        while(Serial1.available()>0){
            incomingByte=Serial1.read();
            if(Buf=='R'){
                for(int i=0;i<22;i++){
                    CurrentID[i]=incomingByte;}
                Serial.println(CurrentID);
                Buf='0';
                Serial.println("Tag Read");
                check=1;
            }else{
                Buf=incomingByte;
            }
            //Serial.println(CurrentID);
        }
    }
    if(check=1){
        displayTime();
        displayID();
        displayTemp();
    }
}

```

```

    check=0;
}

}

```

```

//-----
//-----
//-----
/**displayID_function**
//-----
//-----
//-----

```

```

void displayID(){

    for(int i=0; i<16 ;i++) ID[i]=CurrentID[i+5];
    File dataFile = SD.open("datalog1.csv", FILE_WRITE);
    if(dataFile){
        dataFile.print(CurrentID);
        dataFile.print(',');
    }
    dataFile.close();
return;
}
//-----
//-----
//-----

```

```

//-----
//-----
/**Time_Display_Procedure**
//-----
//-----
void displayTime(void) {
    uint8_t r[8];
    //if time ds1307 is not functioning
    if (!readDS1307(0, r, 8)) {
        Serial.println("Read Failed for display time");
        return;
    }
    File dataFile = SD.open("datalog1.csv", FILE_WRITE);
    if(dataFile){

```

```

// month
dataFile.print(r[5],HEX);
dataFile.print('/');
//day
dataFile.print(r[4],HEX);
dataFile.print("/20");
// year
dataFile.print(r[6],HEX);
dataFile.print(' ');
dataFile.print(Ddd[r[3] < 8 ? r[3] : 0]);
dataFile.print(' ');
// hour
dataFile.print(r[2],HEX);
dataFile.print(':');
// minute
dataFile.print(r[1],HEX);
dataFile.print(':');
// second
dataFile.print(r[0],HEX);
dataFile.print(",");
dataFile.close();
}
}
//-----
//read hex input for time stamp buffer
//-----
//-----
uint8_t hexRead(uint16_t &v) {
  uint16_t n = 0;
  while (!Serial.available());
  while (Serial.available()) {
    uint8_t c = Serial.read();
    n <=< 4;
    if ('a' <= c && c <= 'f') {
      n += c - ('a' - 10);
    }
    else if ('A' <= c && c <= 'F') {
      n += c - ('A' - 10);
    }
    else if ('0' <= c && c <= '9') {
      n += c - '0';
    }
    else {
      Serial.println("Invalid entry");
      return false;
    }
  }
  delay(10);
}

```

```

    v = n;
    return true;
}
//-----
//-----
//-----

//-----
//-----
//-----
/**displayTemp_function**
//-----
//-----
//-----

void displayTemp(){

tempReading = analogRead(tempPin);
    // converting that reading to voltage, which is based off the reference voltage
    float voltage = tempReading * aref_voltage / 1024;
    float temperatureC = (voltage - 0.5) * 100;
    // now convert to Fahrenheit
    float temperatureF = (temperatureC * 9 / 5) + 32;
    File dataFile = SD.open("datalog1.csv", FILE_WRITE);
    if(dataFile){
        dataFile.println(temperatureF);
        Serial.println(temperatureF);
        dataFile.close();
    }
    return;
}
//-----
//-----
//-----

```