**Wildlife Telemetry Receiver Network System**

**A Design Project Report**

**Presented to the School of Electrical and Computer Engineering of Cornell University**

**in Partial Fulfillment of the Requirements for the Degree of**

**Master of Engineering, Electrical and Computer Engineering**

**Submitted by peidong qi**

**NetID: pq32**

**MEng Field Advisor: Joe Skovira**

**MEng Outside Advisor:** [**Julian Kapoor**](https://plus.google.com/u/0/109696811858379741054?prsrc=4)

**Degree Date: January 2018**

**Abstract**

**Master of Engineering Program**

**School of Electrical and Computer Engineering**

**Cornell University**

**Design Project Report**

**Project Title:** Wildlife Telemetry Receiver Network System

**Author:** Peidong Qi

**Abstract:**

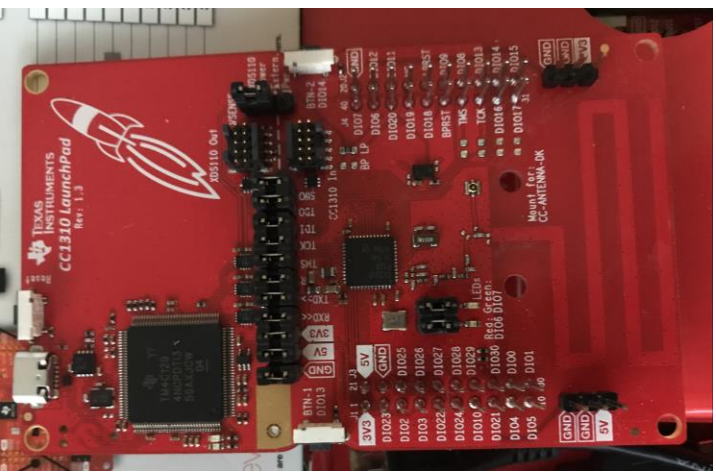
A major challenge for wildlife radio tracking is in obtaining high-accuracy position data of multiple individuals in a population. Systems currently exist that use multiple antennas to create an estimate of location based on the received strength of a signal from three or more receiver towers. However, these systems require bulky and expensive receiver modules, resulting in fairly imprecise location data. Our goal is to create a small (350mg), energy efficient radio tag capable of communicating with the receiver network. For this project, we propose a Angle of Arrival (AOA) based automated radio telemetry system. The basic components of the direction finding (DF) system will use -3 dipole antennas to receive radio frequency (RF) packets from radio tags transmitting at a carrier frequency of ~150 Mhz. The other aspects of this DF system include a low noise amplifier, mixer, and local oscillator (LO) to boost the received signal and mix it with the LO, an RF demodulator to enable the estimation of differences in phase between the received signals, and ADC ICs; they are all fairly typical components of this kind of system. Our system will use the CC1310 wireless-microcontroller (MCU) attached to Raspberry Pi’s to collect the RF data and run the triangulation algorithm. We present a low cost, weight, and power system can operate in short range (100-300m), cluttered environments with high spatial accuracy (~5m) triangulation results for approximately 50 transmitters.

**Accomplishments**

We have worked with the Lab of Ornithology. The main purpose of this project is to track extremely small fairywren. The research group contains five members: one Postdoctoral researcher from the Lab of Ornithology which is our team leader, two MENG ECE students, one senior ECE student, one Junior ECE student. For this project, each one has different work distribution. For my part, I mainly focus on communication between CC1310 and Raspberry Pi. However, we all have contribute to find direction of this project. For each weekly meeting, we discussed the base system design of this project.

For my part, I have found an efficient way to make the communication. I was able to let the CC1310 to send to string to Raspberry Pi.

At the beginning of this project, we have decided to use CC1310 to be our transvers and receivers. Since the CC1310 is very new to this team, we all need to learn how to use this board. We have took serval weeks to get familiar with CC1310.



**System Design**

In a wide range of biology, the ability of free living (or wild) living organisms to exercise, physiology and behavior is proved to be a major stumbling block. Traditional monitoring techniques, such as color bands, stable isotopic analysis, etc., provide extremely rough information about animal behavior in time and space. However, the emergence of radio telemetry of wildlife has fundamentally changed the way biologists study animal movements. Unfortunately, the telemetry of wildlife still suffers from some major setbacks that make it impossible to solve some of the most important questions about animal behavior. Most importantly, one of the major challenges for wildlife radio tracking is to obtain high precision position data for more than one person in the population. Most of the tracking techniques used by wildlife biologists now use 1-3 researchers to track animals on foot (or through vehicles). This method usually results in a few people's low precision position information in a relatively short time. A better way is to automatically track multiple individuals in the receiver network on the fixed tower. There are such automatic systems, but there are many setbacks, including low precision of space information, high cost and low power efficiency.

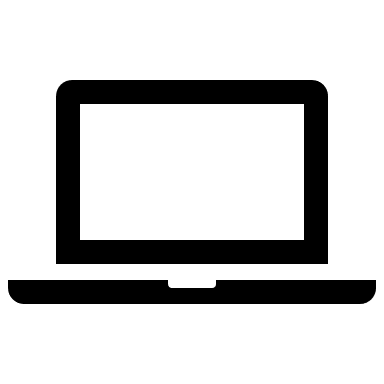
My part of this project is to build base stations which can receive the signal and transfer it to Raspberry Pi. The Raspberry Pi will process and store the all the signals, then send the processed signal to main work station. The main station can use those signals to locate the birds. The problem we have on this part is we need to determine which communicate is best for our project. There are three communication methods: Uart, SPI, I2C. Our initial design was to use I2C to make the communication. The advantage of I2C is we can use more than one master in the electronic circuit design. I2C supports 100 kbps, 400 kbps, 3.4 Mbps. Some variants also supports 10 Kbps and 1 Mbps. In our design, we only need one master device and one salve device for each base station. And I2C need more complicated hardware design. After did some research on I2C, we decided not to use it. Then we have moved on SPI. The advantage of SPI is the transfer rate is fast. SPI have the fastest transfer rate in those three communication methods. Maximum data rate limit is not specified in SPI interface. Usually supports about 10 Mbps to 20 Mbps. However, the SPI require the most complicated hardware connection. It required four pins. And software set up is complicated. After discuss the rough transfer rate of CC1310 and Raspberry Pi, we decided to Uart first and check if it fit our system. The advantage of Uart is easy to set up. The hardware set up is less complexity. And the TI already provide us a sample project for the Uart communication. Since we only need two devices to communicate. As this is is asynchronous communication, data rate between two devices wanting to communicate should be set to equal value. Maximum data rate supported is about 230 Kbps to 460kbps. Without considering the transfer rate, the Uart is the best communication method for our project.

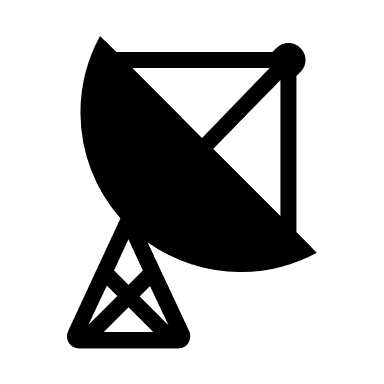
**Design and implement**

For the initial system design, we want four base stations to receive the signal from the tags. Those tags will attach with birds. Then those four base stations send the processed data to main station. We use those processed data to locate the bird. The schematic of system is shown in figure 1

Base Station

Base Station

****



Base Station

Base Station

**Figure 1: Automated phase-base telemetry receiver system**

**Result**

At the end of semester, we were able to use uart to transfer string from CC1310 to Raspberry Pi. We have wrote a python program on Raspberry Pi. This program is going to set up the serial port to enable data transfer. We also run the uart project on CC1310. Since we only need Raspberry Pi the read the data from the CC1310, we only need to connect TX pin on CC1310 and RX pin on Pi together. When we run those two program, we were able to read the string from console in Pi. However we didn’t test the data transfer rate of Uart. We also need to test stability and reliability of Uart. We need to come up plan to do the test. My another concern is the Uart may not fast enough to handle the data transfer. If this happened, we will switch to use SPI.

**Time Line for future work**

|  |  |  |
| --- | --- | --- |
| Task | Time line | Period(days) |
| Test the data transfer speed between CC1310 and Raspberry Pi | 2018/01/22-2018/01/29 | 7 |
| Start to build prototype | 2018/01/22 | -- |
| Help to build a locating system with Matlab | 2018/01/25 |  |
| Start to build a base station prototype | 2018/01/25-2018/02/03 | 8 |
| Work on the receiver CC1310 to get locating signal | 2018/02/03-2018/02/23 | 20 |
| Test long distance signal receiving | 2018/02/23-2018/03/23 | 30 |
| Simulation the location of tag on main station | 2018/03/23-2018/04/23 | 30 |
| Test and debug the prototype | 2018/04/23-2018/05/23 | 30 |

**Reference**

**[1] Y. Ma and E. C. Kan, “Accurate indoor ranging by broadband harmonic generation in passive NLTL backscatter tags”, IEEE Trans. Microwave Theory and Techniques (MTT), vol. 62, no. 5, pp. 1249 – 1261, May 2014**

**[2] Y. Ma and E. C. Kan, “Passive ranging by low-directivity antennas with quality estimate”, IEEE MTT 2015 International Microwave Symp. (IMS), Pheonix, AZ, May 18 – 22, 2015**

**[3] M. Hierold, S. Ripperger, D. Josic, F. Mayer, R. Weigel, A. Koelpin, "Low-weight wireless sensor network for encounter detection of bats", IEEE Topical Conference on Wireless Sensors and Sensor Netw., 2015.**

**[4] M. Hierold, S. Ripperger, F. Mayer, R. Weigel, A. Koelpin, "System design for encounter detection of distributed wireless sensors", German Microwave Conference 2015, pp. 382-385, 2015.**

**[5] Texas Instruments, “CC1310 SimpleLink™ Ultra-Low-Power Sub-1 GHz Wireless MCU”,**

**Internet:** <http://www.ti.com/lit/ds/symlink/cc1310.pdf>

**[6] J.Kapoor, “Wildlife Telemetry Receiver Network System”**

**Internet:https://www.ece.cornell.edu/engineering2/customcf/iws\_meng\_projects/uploads/Skovira\_\_Kapoor\_-\_Wildlife\_Telemetry\_Receiver\_Network\_System.pdf**

**Appendices**

**Python code for serial set up**

import serial

ser = serial.Serial(

port='/dev/ttyS0',

baudrate = 9600,

parity=serial.PARITY\_NONE,

stopbits=serial.STOPBITS\_ONE,

bytesize=serial.EIGHTBITS,

timeout=1

)

print "Serial is open: " + str(ser.isOpen())

print "Now Writing"

ser.write("This is a test")

print "Did write, now read"

x = ser.readline()

print "got '" + x + "'"

ser.close()

**Main code for CC1310**

\* ======== uartecho.c ========

\*/

/\* XDCtools Header files \*/

**#include** <xdc/std.h>

**#include** <xdc/runtime/System.h>

/\* BIOS Header files \*/

**#include** <ti/sysbios/BIOS.h>

**#include** <ti/sysbios/knl/Task.h>

/\* TI-RTOS Header files \*/

**#include** <ti/drivers/PIN.h>

**#include** <ti/drivers/UART.h>

/\* Example/Board Header files \*/

**#include** "Board.h"

**#include** <stdint.h>

**#define** TASKSTACKSIZE 768

Task\_Struct task0Struct;

Char task0Stack[TASKSTACKSIZE];

/\* Global memory storage for a PIN\_Config table \*/

**static** PIN\_State ledPinState;

/\*

\* Application LED pin configuration table:

\* - All LEDs board LEDs are off.

\*/

PIN\_Config ledPinTable[] = {

Board\_LED1 | PIN\_GPIO\_OUTPUT\_EN | PIN\_GPIO\_LOW | PIN\_PUSHPULL | PIN\_DRVSTR\_MAX,

Board\_LED2 | PIN\_GPIO\_OUTPUT\_EN | PIN\_GPIO\_LOW | PIN\_PUSHPULL | PIN\_DRVSTR\_MAX,

PIN\_TERMINATE

};

/\*

\* ======== echoFxn ========

\* Task for this function is created statically. See the project's .cfg file.

\*/

Void **echoFxn**(UArg arg0, UArg arg1)

{

**char** input;

UART\_Handle uart;

UART\_Params uartParams;

**const** **char** echoPrompt[] = "\fEchoing characters:\r\n";

/\* Create a UART with data processing off. \*/

**UART\_Params\_init**(&uartParams);

uartParams.writeDataMode = *UART\_DATA\_BINARY*;

uartParams.readDataMode = *UART\_DATA\_BINARY*;

uartParams.readReturnMode = *UART\_RETURN\_FULL*;

uartParams.readEcho = *UART\_ECHO\_OFF*;

uartParams.baudRate = 9600;

uart = **UART\_open**(Board\_UART0, &uartParams);

**if** (uart == NULL) {

System\_abort("Error opening the UART");

}

**UART\_write**(uart, echoPrompt, **sizeof**(echoPrompt));

/\* Loop forever echoing \*/

**while** (1) {

**UART\_read**(uart, &input, 1);

**UART\_write**(uart, &input, 1);

}

}

/\*

\* ======== main ========

\*/

**int** **main**(**void**)

{

PIN\_Handle ledPinHandle;

Task\_Params taskParams;

/\* Call board init functions \*/

Board\_initGeneral();

Board\_initUART();

/\* Construct BIOS objects \*/

Task\_Params\_init(&taskParams);

taskParams.stackSize = TASKSTACKSIZE;

taskParams.stack = &task0Stack;

Task\_construct(&task0Struct, (Task\_FuncPtr)echoFxn, &taskParams, NULL);

/\* Open LED pins \*/

ledPinHandle = **PIN\_open**(&ledPinState, ledPinTable);

**if**(!ledPinHandle) {

System\_abort("Error initializing board LED pins\n");

}

**PIN\_setOutputValue**(ledPinHandle, Board\_LED1, 1);

/\* This example has logging and many other debug capabilities enabled \*/

System\_printf("This example does not attempt to minimize code or data "

"footprint\n");

System\_flush();

System\_printf("Starting the UART Echo example\nSystem provider is set to "

"SysMin. Halt the target to view any SysMin contents in "

"ROV.\n");

/\* SysMin will only print to the console when you call flush or exit \*/

System\_flush();

/\* Start BIOS \*/

BIOS\_start();

**return** (0);

}