Communication from iPhone to Arduino Through the Audio Jack

Thomas Nattestad
Project available at: https://github.com/thomasnat1/iPhoneToArduino

May 7, 2014

Introduction and Overview

The goal of this project was to establish communication between an iPhone and Arudino through the audio jack. Communication throught the audio jack is something that is clearly possible. For example, the standard iPhone earbuds come with a pause button and volume control. In addition, other devices such as the square credit card reader accomplish actually sending useful, more complex data across.
Communication between Arduino and iPhone is useful because the iPhone (or Android) is a great place to store, visualize, and process whatever data you may be collecting with the Arduino. Additionally, interfacing from iPhone to Arduino opens up a great deal more interesting options of what you can control with your phone. Because while the iPhone is relatively easy to program for iPhone, it is much harder to send messages to devices without using more extensive and expensive tools such as Bluetooth. For this particular project, I decided to work on establishing communication from iPhone to Arduino because it is not as common and therefore seemed more interesting to attempt.

Encoding a Message

There are a lot of different ways that people have come up with to encode binary data in signals. Most commonly used are variations of square waves, such as Manchester encoding.
Sadly, we cannot use something that simple for communication through the audio jack. The iPhone audio jack was designed for audio only, and so has odd limitations on what can be sent through. This requires us to be a bit more clever about how we encode our data.
The most widely used and implemented option is something called FSK, which stands for frequency shift keying. As you can see in figure 1 FSK shifts the freqency of the signal to encode either a 1 or 0. From having a encoded stream of 1s and 0s, it becomes easy to encode messages using the ASCII standard alphanumeric encoding scheme.

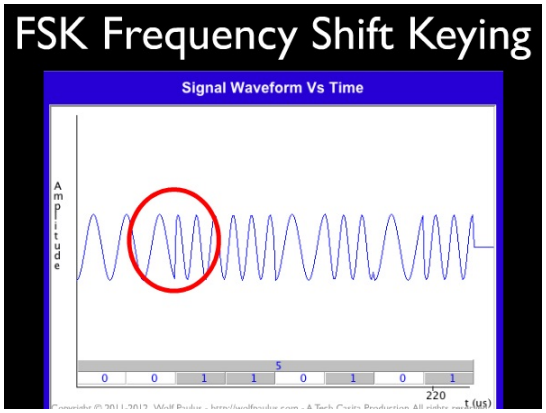

Figure 1: Frequency Shift Keying Explained

Hardware Implementation

The next piece of the puzzle is to figure out the hardware that is needed. The first piece of which was getting an audio jack to plug into the iPhone. There are pre-bundled packages that you can just plug an aux into. I do not have one of those nice packages, so I just grabed an old pair of earbuds and broke off the piece that go in your ear. Within each earbud, there are two wires, one signal and one ground. With a bit of soldering, I was able to plug them into a breadboard. I found a circuit diagram for the easy to use package (Figure 2) and was able to build the circuit, shown in figure 3.

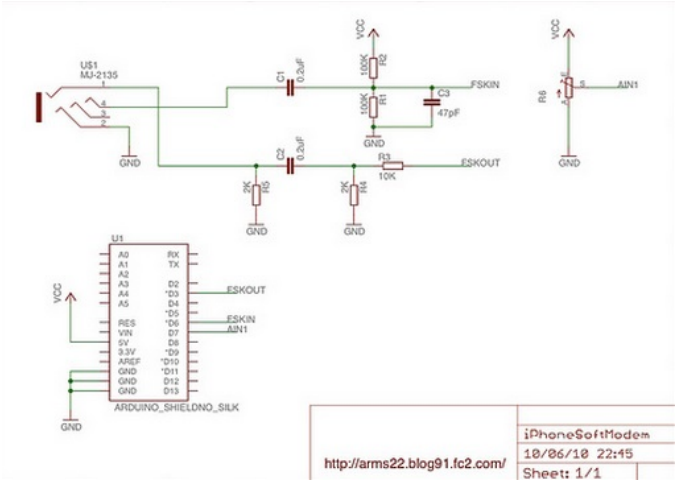

Figure 2: Circuit Diagram for connecting to the iPhone

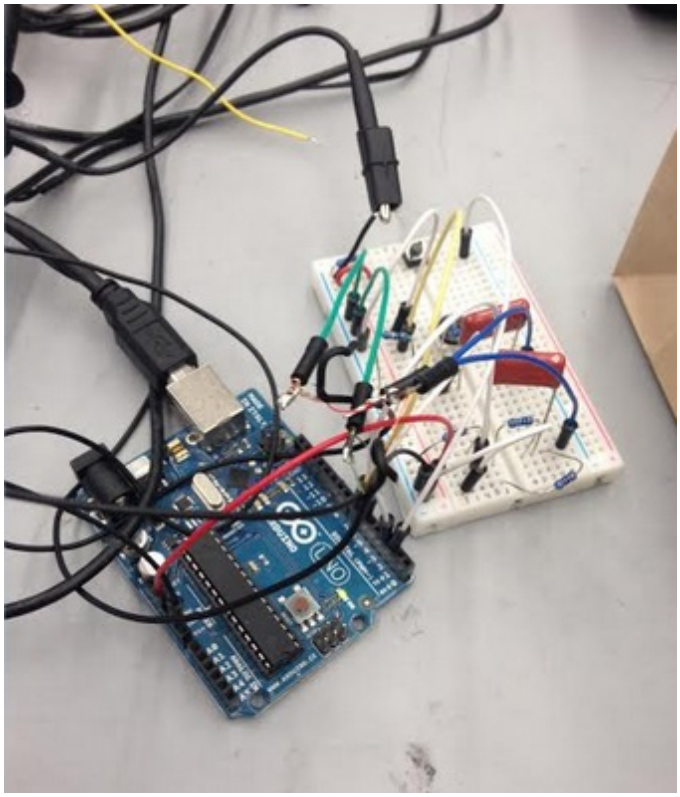

Figure 3: Built circuit

Implementing on iPhone

Knowing how the encoding system works is useful and interesting, but when it comes to actually implementing, it is much easier to find a library that does the crunching for you. Luckily, there is a library called Soft Modem that has both an iPhone and Arduino version. In my case, I was able to grab a github project that included an iOS app that could send encoded hex numbers as sound. I was able to tweak the program to do strip away the pieces I didn't need and change some parameters to suit my needs.
It was very important to test the various components throughout the process, especially since I had gotten most of my information from informal blogs or sketchy chinese websites I had to use google translate to even read. I scoped my circuit as I pushed the buttons on the app and as seen in Figure 4 I got distinct blips of encoded hex numbers.

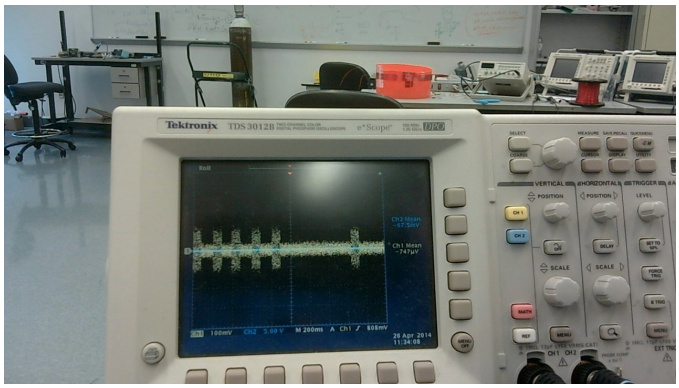

Figure 4: Scoped output of circuit

Implementing on Arduino

The final piece was to actually write the C based arduino code for the communication. The code for this is fairly simple. It interfaces with only two libraries. The first is the complement of the soft modem library used on iPhone to receive communication. One surprisingly hard piece of this code was just figuring out which Arduino pins they were expecting the signal from, which required some digging through their code. The second library is bytebuffer, which I feed the individual bits that is read from the signal and then request packets of hex numbers. I compare these values to those I send from the iPhone and match up the appropriate buttons. Finally, we see in figure **??** the output on the Arduino as I push the buttons in the order seen in the figure.
For now, I do not do anything with the button presses other than output them through the serial. It would be fairly easy, however, to do all kinds of interesting things with this setup. You easily send any hex code from the iPhone and write simple if statements on Arduino to check when those values are triggered.

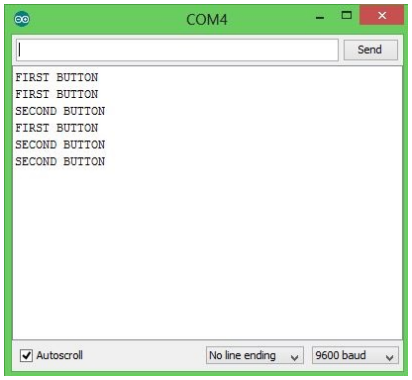

Figure 5: The serial output from Arduino as buttons are pushed

References

1. http://www.creativedistraction.com/demos/sensor-data-to-iphone-through-the-headphone-jack-using-arduino/
2. http://www.slideshare.net/wolfpaulus/android-arduino-and-the-headphone-jack
3. http://forum.arduino.cc/index.php/topic,19648.0.html
4. https://github.com/9labco/IR-Remote