Documentation for the Project Sound Based Transmission of Data

Jordan Eichner

9. Februar 2017

Inhaltsverzeichnis

Kapitel 1

Project Summary

1.1 Motivation

Wireless communication relies most of the time on creating a electromagnetic wave. This can cause interference in certain ares, there fine tuned istruments are used. Another problem exist within buildings, that use thermal isolation, that is not fit for allowing electronic signals to pass through. Hence this project will take a slightly different approach at wireless communication by using sounds. As a mechanical rather than an electrical wave, we don't create interference in electronic devices and sound behaves differently in regard to reverberation. The project's target is to implement a wireless adapter on a microcontroller.
As an example the concept of the modem, a device made to transmit data via telephone cables.
Our Hardware consist of a Teensy 3.1 with a microphone and a speaker, one for each Sender and connceted to a PC via a USB cable. I will create a new protocol for audio transmission, as well as communication between Teensy and PC.
This Project is only a proof of concept and was reduced to a minimum 2 device network with minimal range due to Hardware limitations after an initial possible upscaling solution, which included manual maximum sending frequency, number of channels, minimum channel frequency and much more: The Microphone can't pick up the signals, if the source is not close enough and the Teensy has not enough processing power to analyze properly with high sampling rates, without the sample buffer becoming full, because calculation and serial communication are taking to much time.
This setup also requires a more or less noise free enviorment(i.e. no whistling, loud speech) and lossfree transmission is not guaranteed, because there is no checksum or confirmation process, which could induce a second send.(This is due to limited speed of transmission) There won't be a PC to PC protocol to manage channel reservation, identification of transmitted data(i.e String or Image) and the minimum transmission speed will be awully slow if needed.
Instead of the approach of amplitude modulation, which is used in modems, this project will be using frequency keying, which means, that certain frequencies are linked with a symbol(i.e. low bit). The transmission will be continous sine waves of these specific frequencies.
The inital idea involved the usage of a fast fouier transform algorithm to scan transmission for certain frequencies, but after that proved to be a) time expensive and b) resulted in a heap of seemingly random data, everything was reduced to a much lighter analysis, that is explained in the documentation. It constructs the ideal sine wave for each point in the sample and for each frequency and compares that to the other points of the sample.
The PC Teensy communication is a simple protocol, where the Teensy is a slave, that does sampling, analysis and interpreting the signals, while the PC handels general operation of the Teensy. This

includes stopping and starting of general operation, choosing the channel for own transmission. The PC also catches and handels errors thrown from the device and gives some manual control over certain aspects of the Teensy, see documentation for further details.

Kapitel 2

Protocol of Audio Transmission

There are 2 main ways to transmitt Information via Sound. The first is the keying of symbols to amplitudes. This is hard to produce on Teensy, as there is no good method to control volume of our speakers and the microphone is not very good at picking up quiet noises. The other method is the keying of symbols to specific frequencies. For this proof of concept I limited myself to 2 devices and therefor 2 channels of transmission. Each Channel had 3 Symbols and therefor 3 frequencies.

1. Low Bit, the Teensy would assemble 8 bits to a Byte before sending it to the PC.
2. Identification, this symbol seperates to data bits, as well as reserving the channel for the Teensy.
3. High Bit, see low bit.

As described in the section about the signal analysis we divide the signal into sample of a specific length. Each signal is sended for double this time window. This way the Teensy either finds:

1. B and I or I and B. B is either low or high
2. B and B.
3. I and I.

If we drop every second sample, the Teensy receives additional time for analysis as well as ensuring, that it never picks up a data bit twice. Based on the cycle a Teensy is operating in, the Teensy need to adjust its limits(see L_2) as its pick up a signal or switch its cycle of dropping(here done via a serial command).

Kapitel 3

Serial Protocol

The following is used for communication between Teensy and the connected PC as a controller. To synchronize the flow of transmission the Teensy awaits the first byte from the PC, before starting its services.

3.1 To Teensy

All communication is done by directly writing bytes.

1 Stops the device.

2 Restarts the device.

3, X Start Sending on Channel X.

4, X Send Byte X.

5 Switch Cycle of dropped Samples.

In the first iterations this protocol handeld much more stuff, like frequencies, number of channels, but it got thrown out, to simplefy the proof of concept.

3.2 To PC

The Teensy uses its internal *println*() function. This output is in utf-8. The Arguments are seperated by ','.

1,X Found Channel X.

2,X,Y Received Byte Y on Channel X.

3,X Error with Code X occured, see below.

The following are the Error Codes.

1 Sample buffer full

2 Outgoing buffer full.

3 Serial buffer full.

Kapitel 4

View of Teensy

4.1 Hardware

- Arduino KY-038 Microphone sound sensor module
- Teensy 3.1
- A speaker system

4.2 Software

The Teensy is programmed via the Arduino IDE with its specific libary Teensyduino.

4.3 Signal Analysis

4.3.1 General Theory

As specified above the Teensys communicate via sound within a given spectrum, using different frequencies for their 3 signals (hold,high and low). The Nyquist–Shannon sampling theorem states, that we need a sampling rate of at least double the to be sampled signal frequencie to use it for later applications like FFT. So we have a maximum sending Frequencie called $F_{max}hz$ and a sampling frequency $F_{sampling}hz = 2F_{max}$. FFT divides its input into a number of n bins, each representing $\frac{F_{sampling}}{n}hz$. This input needs a length of $2*n$. To take $2*n$ samples we need $\frac{n*2}{F_{sampling}}s$. This is the time window in which we need to analyze a sample and doing other stuff, like serial communication and audio output. Although I no longer use FFT, I still take samples of $2*n$ to analyze them, because of this time window.

4.3.2 History

FFT

At first it was planned to use fast fouier transformation to search in a measured sample for a signal. In a first try I made use of a FHT libary written for the Ardunio Framework. After I constructed the surrounding framework for sampeling and pC communication, it turned out that the libary was not compatible with Teensy 3.1 Hardware because of different asembler specifications. So it got thrown out and replaced with KISSFFT, a libary written in C, that at least looked functional on

Teensy. A few weeks were spend to fit the runtime of FFT into the time window of the sampling. Everything seemed ok, until the first test, where the framework started to act up as FFT started to produce complety random results. After some tweaking with no results, I had to throw it in the gargabe and looked for a much simpler solution.

Analysis via Trigonomy

So I sat down and looked at the pure Input from Microphone. Its sensitivity was really low and sound was only picked up if it was extremly loud or the source of the sound extremly near, like 1cm and less. This could I use, since if a sine wave was applied as a signal it were almost identical to its measured output. The follwing algorthim were conceived to scan an input $(P_0 \dots P_n)$ wave for specific frequency f_t.

1. Calculate the offset V_f. This is done by scaling the sine wave of the signal on $f = 1hz$ and dividing by the window between two samples.
   (a) $D_f = 1/f_t$ and $D_s = 1/F_{sampling}$
   (b) $\frac{D_s * 2\pi}{D_f} = V_f$
2. Map all points P to the range of $[1 \cdots -1]$
3. For every point we calculate their respective multiple of π within a regular sine wave Q_i, using $Q_i = \arcsin(P_i)$.
4. Compare a ideal sine wave of the frequency f_i, constructed from every point P_i , with all other subsequent pooints. For an explanation of L_1 see below.
   (a) $S_f = \frac{\sum_{i=0}^{n} M_i}{n}$
   (b) $M_i = \frac{\sum_{j=i+1}^{n-i} N_j}{n-i}$
   (c) $N_j = \begin{cases} 1 & abs(\sin(\arcsin(Q_i + j * V_f)) - Q_j) < L_1 \\ 0 & \text{sonst} \end{cases}$
5. S_f represents now the resemblance between a ideal sine wave of the frequency f_t and our sample.

The only thing remaining is to find the limit L_2 for S_f to identify a clear hit. This limit is hardware and setup specific.
L_1 from above is the error we would allow within our measurements and because of floating point numbers. For that we just half the minimal difference between two V_f for all target frequencies.

Runtime

- Step 1 can be done at initialisation, before we take samples and is irrelevant.
- Step 2 and 3 have a Runtime of n.
- Step 4 is done in n^2.

The lookup for sin and arcsin is really costly for teensy, because of that we only construct sin waves from every 9^{th} sample and we are ignoring all channels for identifications.

4.4 Sampling

This module consists of 2 parts:

- An IntervalTimer that comes from the Teensy Libary and is used to take a sample every $\frac{1}{f_{sample}}s$
- A Ringbuffer for the IntervalTimer to store and analyzer in the main function to get, without disturbing each other.

4.5 Sound Generation

We use a second IntervalTimer that produces a sine wave at the given frequency f, by switching through the different values for a digital write at a fixed interval $\frac{1}{f}s$.

4.6 Communication with PC

For the protocol see above.
In the main routine we handle ingoing data directly as they come.
For outgoing data we use a ringbuffer to store it, while in the main routine we send its content ensuring that errors within an interrupt a full serial buffer doesn't slow down it's progress.

4.7 Specification for Sound Transmission for this Project

We have 2 channels. Using the following the frequencies (in this order: $L_0, I_0, H_0, L_1, I_1, H_1$): $f_0 = 600hz, f_1 = 680hz, f_2 = 760hz, f_3 = 840hz, f_4 = 920hz, f_5 = 1000hz$. We have a sampling rate of $F_{sampling} = 4400hz$. $L_1 = 0.03$. Sample Size: $n = 256$. Error Limit: $l_1 = 0.03$

Kapitel 5

View of PC Side

The PC uses a simple Python(v.3.6.0) program, using the libaries pySerial and TkInter.

- The program runs on 2 threads: $message_{grabbing}, message_{handling}$.
- Data between the first two thread is done via a Queue.
- In the main thread runs the GUI loop.
- Its content is updated by $message_{handling}$.
- Commands for the Teensy are also Queued and then sended at the end of a $message_{handling}$ loop.
- The program shuts down, when the GUI is closed.

Kapitel 6

History of Project

12.11.16 Creation of the wiki and first tests of the Teensy Hardware(Serial, TimerInterval, Tone, analogRead). Single elements seem to work properly, although the microphone produces a lot of noice, hope that doesn't break my legs in this project.

29.11.16 First framework of Tone and IntervalTimer with Serial for further test. Note to myself the Serial Interface is not fast enough to dump all samples to the PC, so we can do stuff here.

8.12.16 First libary for data processing(ring buffer), as well as more on the theory about FFT, like sampling rate, speed of transmission, sampel size.

13.12.16 Found a FHT algorithm for the sound analysis. Moved Stuff into Github and tried without success to conncet lab PC with Github... Working on Wiki Page.

20.12.16-27.12.16 Sickness and Holidays, no progess...

10.1.17 Wrote the whole first framework for Teensy and PC.

17.1.17 Note for myself: Don't try to make stuff shorter by using generic libaries, had to rewrite the two ringbuffers as the generic implementation killed the Teensy program.

24.1.17 Needed to search for a new FFT libary, as the original FHT doesn't run properly on the Teensy, because of a different processor architecture to regular arduino boards.

31.1.17 Finally managed to get KissFFT running on the Teensy without having to stop the sampling because of a full sample buffer.

5.1.17 After a few dozens tweaks at the software and some test doing FFT on the PC, which would not be a permament solution(see note from 29.11), FFT doesn't produce expected results. Beginning to make a simple solution with Trigonomy.

8.1.17 Implemented a smaller solution for the analysis and copied documentation for this project from my numerous hand and PC notes into a central file. As I was making final adjustments my micro started hickups. Because I didn't saw the need to get the second device with micro, without a functional code, I have no backup until tomorrow to test the complete framework and the last fixes of the implementation. Created a presentation for this project.