Spring 2018 CS120 Project 1. Building an Acoustic Link

Handout Date: Mar. 1, 2018 Due Date: Mar. 28, 2018

(10 points + 6 points)

Please read the following instructions carefully:

- This project is to be completed by each group **individually**.
- Submit your code through Blackboard. The submission is performed by one of the group members.
- Each group needs to submit the code **once and only once**. Immediately after TAs' checking.

Overview

The goal of this project is to set up a link (not necessarily 100% reliable) between your devices through acoustic signals. A high-level architecture of the program may look like:

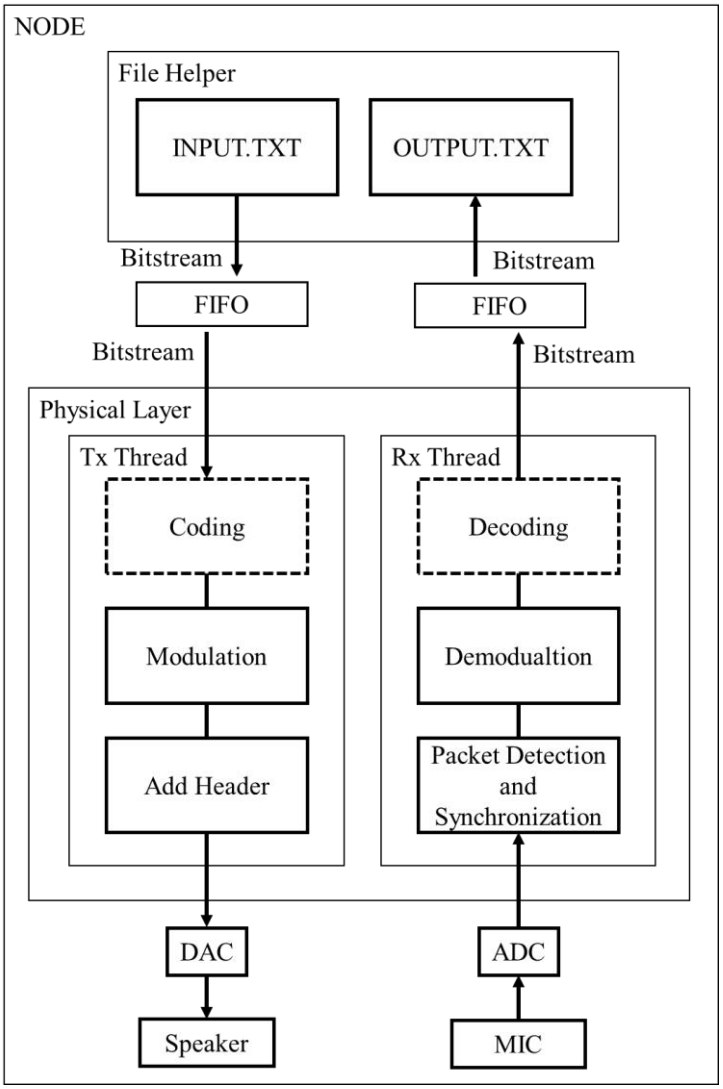

Figure 1 Project1 Overview

Part 1. (3 points) Understanding Your Tools

Different operating systems have different architectures for media I/O, but their usages are quite similar. In order to convey bits through acoustic signals, you must be able to correctly send and receive through the media interface.

For programming in Java, you can start by referring to the Java Sound APIs[6]. JDK 9 is the latest java version, but JDK 8 is recommended. Remember Java Sound APIs are more powerful than we actually need. You should focus on the low-level sound APIs (e.g. sampling sound and transmitting samples).

In this part, your task is to correctly manipulate your devices' ADC and DAC (i.e., it's about how to play digital samples through speaker and how to record sound from microphone). The playing and recoding should be able to perform simultaneously. I suggest you to use two independent threads and buffers to control the ADC and DAC respectively.

Checkpoints:
The group provides one device: NODE1.
CK1(1.5 points). NODE1 is able to record the voice from TA (10 seconds) and then play the recorded signals.
CK2(1.5 points). NODE1 is able to play a predefined sound (any) and record the played sound at the same time. TA may say something during the recording. After 10 seconds, stop playing and recording. Then play the recorded sound.

Tips:
a. Be careful about stereo settings. We only need one track
b. Be careful about the sampling rate of the ADC and DAC
c. You can record sound into a file and then use Matlab to help you debug

Part 2. (1 points) Generating Correct Sound

Knowing how to play and recode sound does not help in generating a correct sound signal. The later task is more about calculation and matching sampling rate. Suppose the sound you want to generate is f(t), if the DAC uses sampling rate fs=44kHz, the digital samples filled in the DAC buffer are calculated by f(0), f(1/fs), f(2/fs), …, f(n/fs).

Checkpoints:
The group provides one device: NODE1.
CK1(1 points). NODE1 is able to play the signal: sin (2pi 1000 t) + sin (2pi 10000 t)

Part 3. (5 points) Convey Your First Bit

Modulation. You can choose one of the modulation methods from lectures slides or whatever you like (from wiki or reference links). There is no best choice for all situations, but there exist good ones for certain situations. In this description, I use phase modulation (PSK) as the example (it may not actually work for this project). The first thing is to determine the duration of each symbol/bit (i.e., the bandwidth). Suppose phase 0 is used to represent 0 and phase pi is used to represent 1, and the carrier wave is at 10kHz. If you want to achieve 1kbps, then the duration of each bit is 1/1000 s. There are 10 cycles of the carrier wave in this duration:

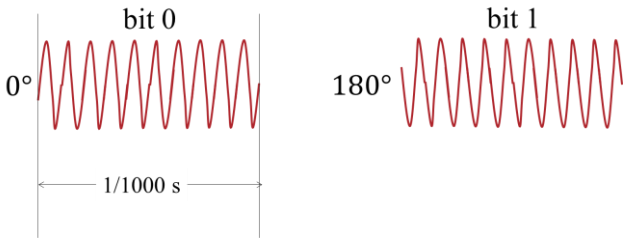

Figure 2 PSK Example

Now you know the modulated signals. The next step is to translate the signals into the digital sample so that DAC can understand it. You have finished this step in Part2.

Adding a Header. The major function of a header of each frame is to help the receiver to find out the start of the frame and also synchronize the clock. The header could be a special bit pattern, which can be added to the bit stream before modulation. More often, the header is a manually-defined special wave pattern and concatenated to the output of modulation. After adding the header, samples that you are going to fill into the DAC may have the following structure:

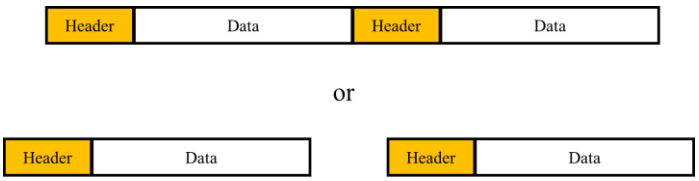

Figure 3 Adding a Header

Packet Detection. When the device receives enough samples, the first thing is to determine whether there is something transmitting and where is the start of the received frame. Correlating your predefined header with the received samples can give you very high confidence on the existence of new frames and its rough start point. So a unique header may help a lot.

Synchronization. Once the start of a new frame is confirmed, the next thing is to find the accurate boundaries of the symbol/bit. You can develop your own way. One suggested way is to use the correlation value of the header as the indicator. Larger correlation value normally indicates higher accuracy in alignment.

Demodulation. The demodulation is the inverse process of modulation.

Tips:

a. Take a look of your modulation scheme. If the bit/symbol are statically mapped to a certain wave. You may consider using LUTs to reduce the real time calculation overhead.

b. You can determine how long a frame should be, but you'd better do not use a very long frame. This is because the ADC/DAC rates of different devices are slightly different (i.e., frequency offset. For a one-second recoding, the number of samples from different devices may have difference up to ~100).

c. You can determine how long a header should be, but you'd better do not use a very short header. Speaker takes time to warm up (see ringing effect [1]).

Checkpoints:

The group provides two devices: NODE1 and NODE2

CK1(6 points). The TA provides a TXT file "INPUT.txt" which contains 10000 "0"s or "1"s. NODE1 sends bits according to this file. NODE2 stores the received bits into a TXT file "OUTPUT.txt". During the transmission, TAs keep quiet.

The transmission must be finished within 15 seconds.

<15s	-0%
>15s	-100%

TAs compare the difference between INPUT.txt and OUTPUT.txt through a tool similar to "diff":

<80%	-100%
80% to 99%	-20%
>99%	-0%

Part 4. (1 point) Error Correction

The current link is fragile to noise. You can add redundancy in your bits to resist error caused by noise (we call this procedure as coding). There are many coding schemes. Choose one from [5] (Convolutional code and Fountain code are suggested).

Checkpoints:
The group provides two devices: NODE1 and NODE2
CK1(1 points). The TA provides a TXT file “INPUT.txt” which contains 10000 “0”s or “1”s. NODE1 sends bits according to this file. NODE2 stores the received bits into a TXT file “OUTPUT.txt”. During the transmission, TAs will clap their hands for four times.

The transmission must be finished within 30 seconds.

<30s	-0%
>30s	-100%

TAs compare the difference between INPUT.txt and OUTPUT.txt through a tool similar to “diff”:

<100%	-100%
100%	-0%

(Tasks with "Optional" tag are optional tasks. The instructor is responsible for checking and grading the optional tasks. Contact the instructor to check if you have finished one or more of them)

Part 5. (Optional + 5 points) Higher Bandwidth

The Bandwidth of your acoustic link is limited by many factors. A big one is the echoes. The echoes (i.e. signals from multiple paths) blur the boundary between the symbols/bits. Handling echoes is a challenging task. One way is to estimate the property of the echoes and try to cancel it out (see equalization [2]). Sometimes we can leverage advanced modulation (see OFDM[3]) to reduce the impact of the echoes. Some guys have already given a good example on how to implement OFDM in acoustic channel [4]. I suggest all of you take a look at it. The article may help your implementation even if you are not going to implement OFDM.

Checkpoints:
The check routine is similar to Part3.

Part 6. (Optional + 1 point) Need for Speed

Do not use float, double, division, and multiplication in the calculation of your program.

Tips:
a. Use integer and bit shift operations.

Checkpoints:
The check routine is similar to Part3.

Reference and Useful Links

[1] Ringing Effect https://en.wikipedia.org/wiki/Ringing_(signal)
[2] Equalization https://en.wikipedia.org/wiki/Equalization_(communications)
[3] OFDM https://en.wikipedia.org/wiki/Orthogonal_frequency-division_multiplexing
[4] Acoustic NFC http://dl.acm.org/citation.cfm?id=2534169.2486037
[5] FEC https://en.wikipedia.org/wiki/Forward_error_correction
[6] JAVA Sound API https://docs.oracle.com/javase/8/docs/technotes/guides/sound/