**ECEN 192**

**Lab 8 – Speakers and Sound**

**Overview:**

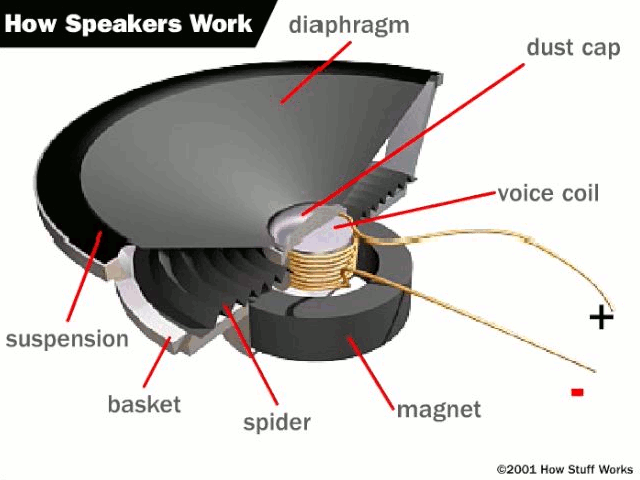
In this lab, you will learn the basics of speakers and frequency generation. You will also learn how we store sound files and what a digital sample is. To conclude, you will explore some basic audio filters, and will have the chance to experiment with a few of these using your y-badge.

**Introduction:**

One of the most important aspects of any computer system is being able to interact with it. Audio is one of the most common forms of human-computer interaction. Think back to the last 24 hours. You likely used computer audio several times to watch a YouTube video, Instagram reel, or even make a phone call.

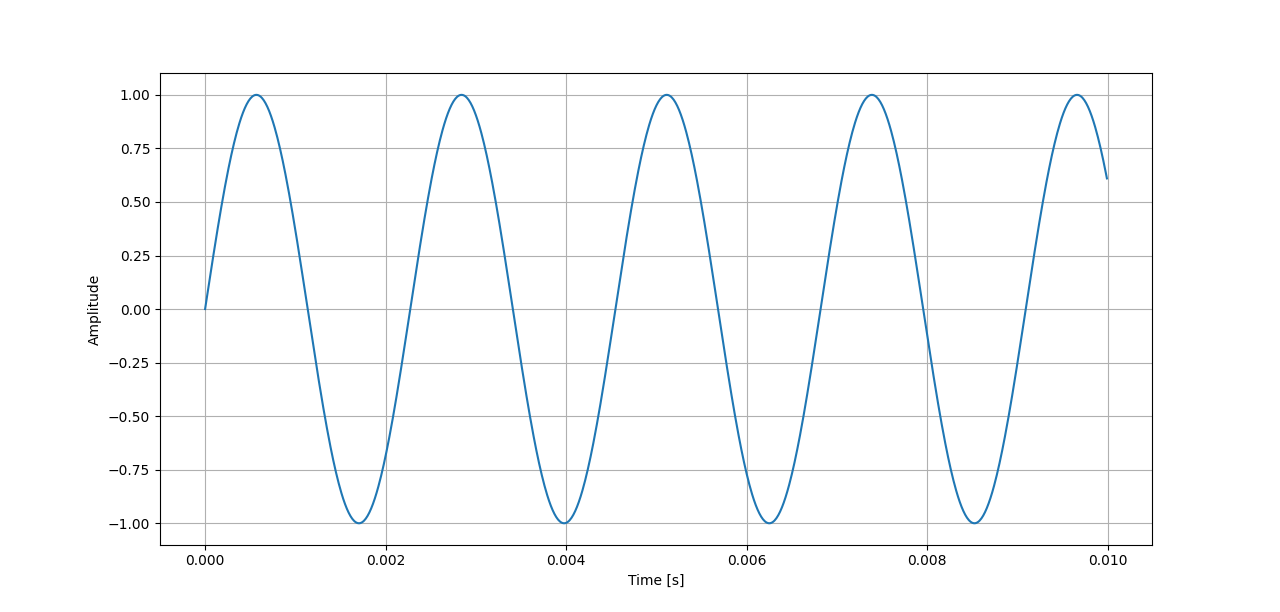
**Speakers:**

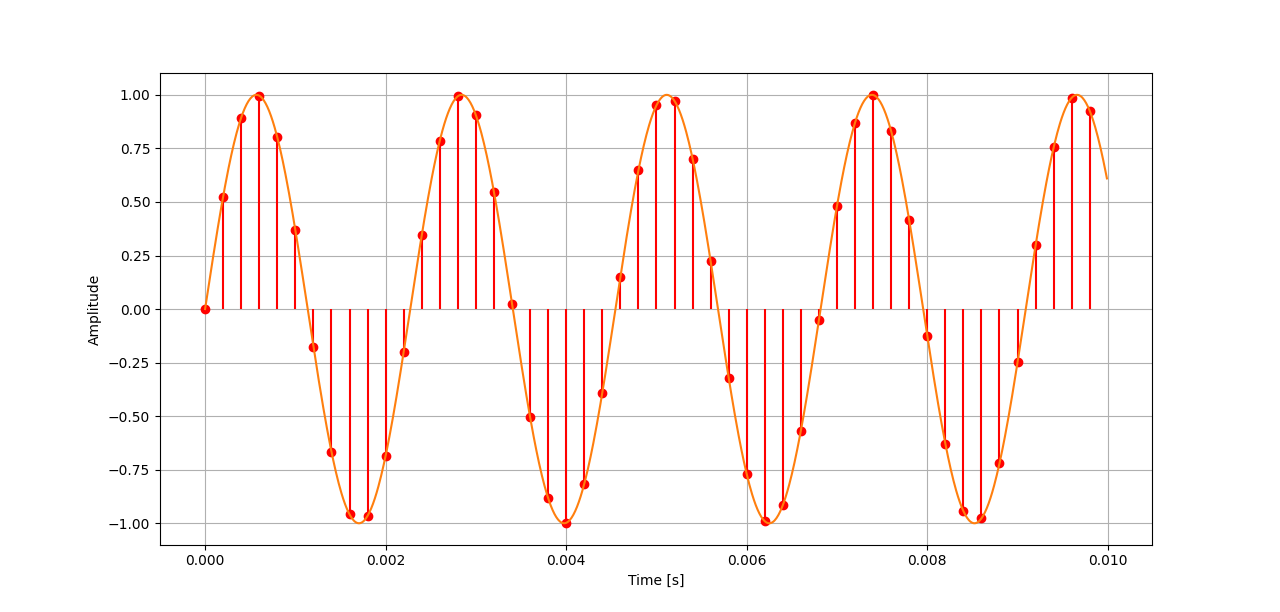
Speakers work by converting electrical signals, called samples, into sound waves that we can hear. This is done through an electromagnet and a diaphragm. As the electromagnet pulls the diaphragm forwards and backwards, it creates pressure waves that are propagated through the air as sound.



An important concept to understand in the world of computer audio is **sample rate**. A sample is a number that tells the electromagnetic and diaphragm in the speaker what position to be in. By changing this number, we can cause the diagraph to move and create a pressure wave in the air. The sample rate is the rate at which we send these numbers to the speaker. The big question then becomes, how high or low should our sample rate be to create good quality audio.

Here is a graph of a wave that represents a tone with the frequency 440 Hz (A4 on the piano).



This wave is continuous, meaning it is defined at every point with no gaps. Computers, however, do not have infinite memory to store these samples, so we must choose when to take a sample. The more samples we take, the better the audio quality. However, there reaches a point where the human ear can’t tell the difference, so we don’t go overboard. Generally, we take 44,100 samples per second (44,100 Hz), though due to computational limitation so the y-badge, we use slightly lower than that. Here is the same tone but how we could sample it.  
These samples are stored in an audio file. The most basic type of audio file is the wave (.wav) file which is what we will experiment with today.

**Lab Procedure:**

1. Explore Tone Generation

First, experiment with generating simple tones. Given a frequency, we can tell the y-badge all the samples it needs to play a single tone. Download the .zip for the lab’s code here. Unzip and open it, as in previous labs. This code is ready to be compiled and sent to your y-badge. Click the upload button. Once complete, the screen should show the current selected frequency. By pressing button 1, you should be able to generate and play a tone at that frequency. You can adjust the current frequency and the associated musical note by turning the potentiometer knob. The knob can be a little sensitive, so you might have to make small rotations to get the note you want.

1. Putting Tones Together

Now that you are able to play single notes, we can string them together to make a song. In your loop() function, comment out the line that says

“””

tone\_gen\_loop();

“””

by putting two slashes (//) in front of it. Then uncomment the line following, which says

“””

play\_notes();

“””

by removing the slashes in front of it. In the code, we already have several songs that you can choose from. They are:

* star\_spangled\_banner
* mario
* star\_wars
* fight\_song

Insert one of these between the parentheses, replacing the phrase “put something here” to have the y-badge play this song when you press button 1. Upload the code and try it. If you are feeling adventurous, you can look at the code to see if there is another secret song programmed in. You might want to look at the file tone\_gen.h. You can also create your own by putting in a string of notes between quotation marks. Check out the table at the end of this document to see the syntax on note entry.

1. Play a Wave File

Playing a song might be interesting, but we want to be able to play any audio file, not just single synthesized notes. On your SD card, you should have several audio files preloaded that we will use today. Insert the SD card if it is not already. Comment out the play\_notes line and uncomment the next, that says

“””

play\_wav():

“””

Upload the code and you will hear your y-badge play *Ode to Joy* when you press button 1*.*

1. Audio Filtering

Have you ever noticed the knobs on stereos that adjust the treble and bass? A very important part of audio processing is being able to apply filters to improve sound quality. We can use filters to increase the bass in a song, remove the high-pitched ringing in a poor audio recording, or even add reverb to someone’s voice.

At the top of the code, you will see a variable named FILENAME. Change the file path to be

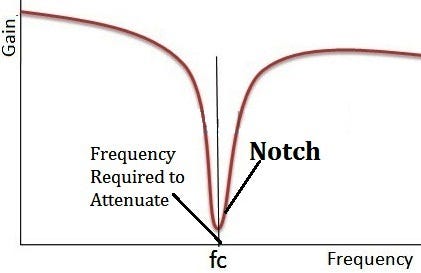
“””

“/audio\_with\_tone.wav”

“””  
Upload the code and play this .wav file. You will hear a single tone and if you listen closely there is a quite voice speaking at the same time. We want to use a filter to remove the tone and understand what is being said.

Now comment out the play\_wav(); line and uncomment the following line that says filters();

There are two main kinds of filters in audio processing, called high-pass filters and low-pass filters. A high-pass filter only lets high frequencies through (i.e., they pass through the filter), while a low-pass filter only allows low frequencies. If we want to remove all the high-pitched ringing from a recording, we can use a low-pass filter. But we can do better with our audio file. The tone being played in the recording is 440 Hz or A4. Combining a high-pass filter and a low-pass filter in a clever way can give us a new kind of filter called a notch filter. A notch filter allows all frequencies through except for a small range, removing a notch from the frequency spectrum where we don’t want it. This graph might help. The y-axis is the gain or volume of the sound, and the x-axis represents different frequencies along the spectrum.



In the paratheses of the filters function, replace the phrase “put something here” with the frequency around which we want to remove, which is 440. Upload this code. When you press button one, you will hear the original sound file. When you press button 2, your board will filter the file and play the new audio. Can you hear what the secret word is? The tone won’t be removed completely because it is so loud in comparison, but you should be able to understand the voice now.

**Further Exploration:**

You were able to play the .wav files we gave you. You can also add your own on the SD card and play them. All you have to do is change the FILENAME at the top of the code to the new filename. The y-badge is only coded to play .wav files with a single channel and a sample rate of 16,000 Hz. Most audio editing tools, like Audacity, can set this.

If you want to play your own song using tones, instead of putting the variable name for the song, insert a string of note instructions between quotes like this:

“””

“C4 E4 G4”

“””

Here is a table for the note notation. It should be noted that the number following the note name does not signify the octave as in most musical notation but the duration of the note.

|  |  |  |
| --- | --- | --- |
| Command | Notation | Example |
| Play a note | Use the note name | “A B C” |
| Set the note duration | Place a number following the note for how many of them go in a measure. (ex 4 would be a quarter note, 8 would be an eight note). If no number is given, the default is 4. | “C8 E8 G4 E2” |
| Sharps or flats | Place a + or - sign after a note for sharp or flat respectively. | “C+ F G+” |
| Set the tempo | Use the letter T followed by the new tempo in beats per minute (bpm) | “T80 A8 B8 C8 T120 A8 B8 C8” |
| Set the octave | Use the letter O followed by the new octave number. The default is 4 and valid ranges go between 4 and 7. | “O4 C4 O5 C4” |
| Rest | Use the letter followed by the duration of the rest | “C4 R8 E4” |
| Dotted rhythm | Place a dot after the note to produce a note duration that is 50% longer | “C4 D4. C4” |