Computer Science 327
Project 1, Part a
C Programming Project
Digital Sound

Notes

Some parts of this project contain mathematical formulas, but nothing more than you have seen in Math 165. This is not a class in digital signal processing, nor the mathematics involved in processing sound with algorithms. However, to create interesting assignments for both engineers and computer scientists, this project contains real-world applications, or approximations of real applications. As you read through this assignment, and its subsequent parts, there are several places where further explanation will be given in the lectures. As always, if something is unclear, please see a TA, me, or ask on Piazza for clarification.

Introduction

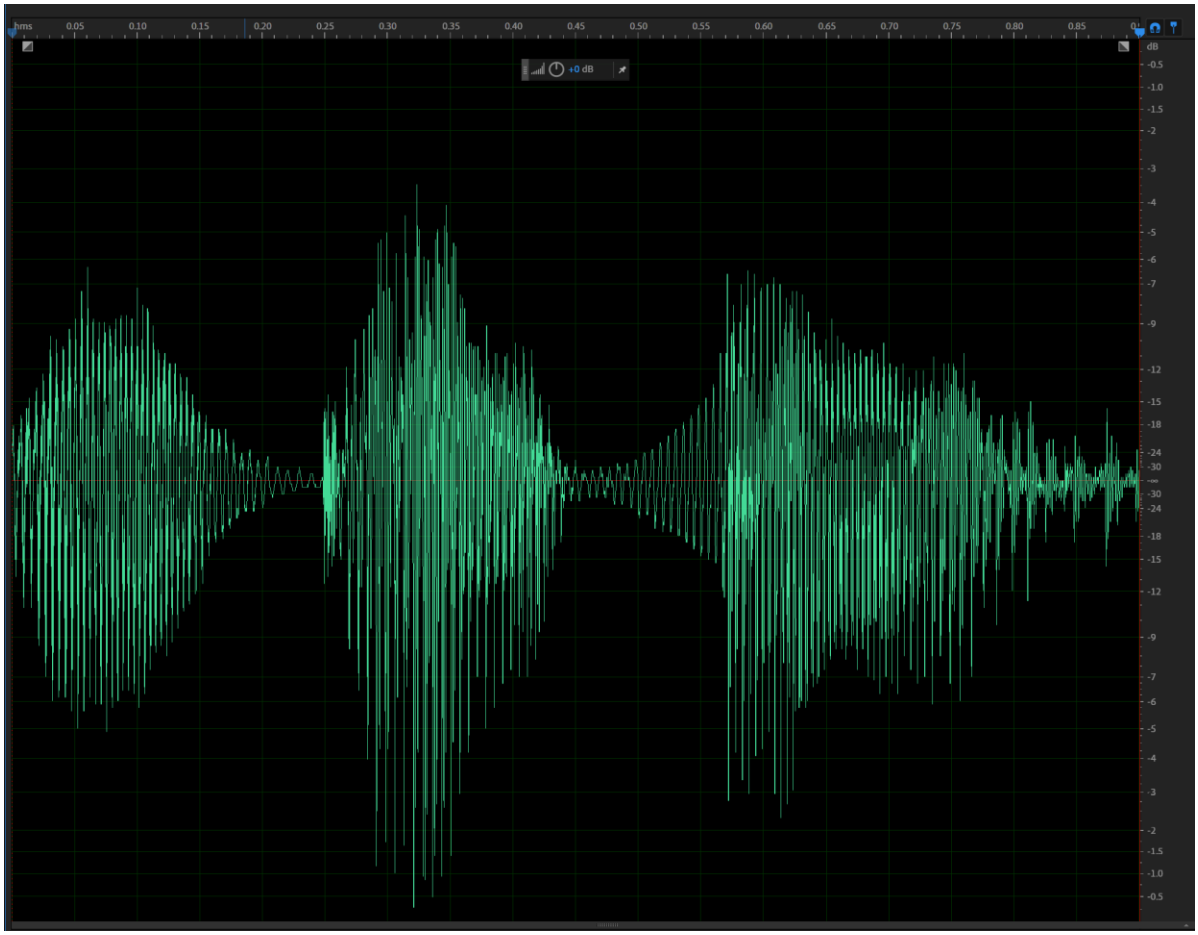
This project utilizes digital sound processing techniques to perform various tasks on small sound samples. Again, you are not responsible for understanding the mathematics behind formulas and algorithms given to you, but you are required to translate these formulas and algorithms into C programming source code and working programs.

Sound Background Information

Sound is simply a wave transmitted through the atmosphere through changing pressure. These waves, or changes in air pressure, are detected inside the ear, and then translated into what we hear. To generate a sound wave, we can use a speaker to vibrate which in turn compresses and decompresses the air around it. Done repeatedly, it produces the changes in pressure that we perceive as sound. Records, (the thing that us older people bought back in the 1960s and 1970s, before compact disks became popular) store recorded sound in the depth of the grove etched into the record. This depth is "read" by a needle that vibrates as it travels along the groove and moves over the changes in depth. The record player then converts this vibration into an electrical signal that is then amplified to drive a speaker, which then produces the sound from the record.

Because sound is a wave, it has a frequency. The higher the frequency, the higher pitch the sound. For example, a single sine wave at 440 hertz (cycles per second) corresponds to concert A and is universally used to tune instruments. In fact there is a Youtube video that is 10 hours of this note https://www.youtube.com/watch?v=xGXYFJmvIvk. The example here uses a sine wave so that it is the only frequency present. However, most sounds are complex and consists of many sine waves and different frequencies, amplitude and phase. In addition, sound is most

useful if it varies over time. (There are not that many one-note songs) So, when we store a representation of sound, we cannot just store the frequencies, amplitudes, and phase of a single note, we must store the sound pressure information continuously over all time that the sound (speech or music for example) is present. Below is an example of an audio wave of a voice speaking.



A digital recording uses analog to digital converters to translate the electrical signals that correspond to air pressure into binary numbers that can be read easily by computers, and signal processors. Note that it is easy to make a sound louder or softer by multiplying all the numbers (samples) of the digital sound data by a constant number. (A number greater than 1 will make it louder, and a value less than one will make it softer. Can you explain why adding a constant value to all the numbers does not change the sound that you hear?) Sound data is often stored on computers in files in a variety of formats, including MP3, WAV, etc. While this project will eventually utilize reading and writing sound in a restricted wav format, this part of the project does not require that.

It is also important to understand that each digital sample of sound is taken at a particular time. For a sequence of samples as is normally the case, it is important that each sample be taken at equal time intervals. This is called the sample rate and is given in terms of samples per second.

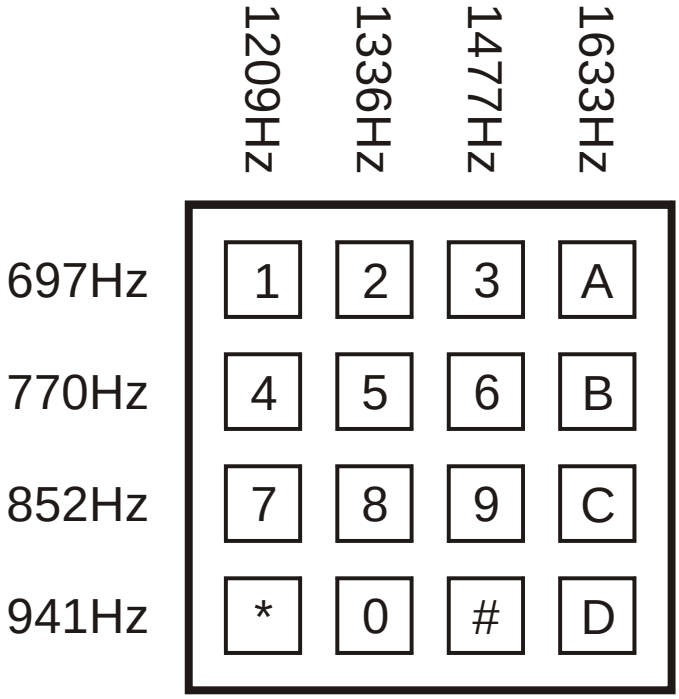
(often called a hertz) For example, the samples above were taken at 16000 samples per second, or 11,000 Hz, or 11 KHz. (kilohertz). A well-known theorem says that the highest frequency you can reproduce with a digital sample is half of the sample rate. Thus, the highest frequency that can be recorded with an 11 kilohertz sampling rate is 5.5 kilohertz. Compact disks that store music have digital data that is stored at 44.1 kilohertz and thus the highest frequency sound that a CD can play is 22 kilohertz. (Most humans can only hear up to about 20 kilohertz.) The example above when viewed over a very short period of time shows how samples are used to create sound. This is shown in the image below. The process of converting samples back to an analog (continuous) wave that can be output to an amplifier and speaker is called reconstruction. This process is somewhat complex and is not discussed here.

DTMF and your phone

DTMF stands for Dual-Tone Multi-Frequency and is most commonly used in telephone dialing and signaling. We begin with an explanation of DTMF. Using 8 different tones (sound frequencies), we can encode 16 different code-words (characters, keys, etc) by utilizing a 4 by 4 array and labeling the rows and columns with the frequencies. Using the following frequencies,

697 hertz,
770 hertz,
852 hertz,
941 hertz,
1209 hertz,
1336 hertz,
1477 hertz,
1633 hertz,

we can create the following array:



It is not surprising that part of this array looks very much like a telephone keypad. In fact, when you press the "1" key on a touch-tone telephone pad, the telephone generates a tone composed of a 697 Hz tone and a 1209 Hz tone. This dual-tone sound is received by the telephone equipment at the other end of the telephone line and is used to determine the number you are dialing, or in more recent years, to navigate phone trees. Creating these "touch-tones" is as simple as adding two sine waves together of the correct frequencies, and scaling the result.

The Assignment (Part a)

1. Write C source code contained in the file "gensnd.c" that contains a function called "gensine" that outputs a sine wave at a specified frequency (in hertz) using a specified sample rate (in hertz), and for a specified time duration (in seconds). These parameters are float type. The output you generate are floating point numbers between -1 and 1, one number per output line. (Note that in a later part of this project this function will be rewritten to return an array, and other functions will be added to convert the samples to integer numbers. In this function, the return type is void and does not return a value. The function writes the desired data to the standard output.)
2. Write a "main.c" program that asks the user to input a frequency, sample rate, and duration, and then outputs a sine wave with those characteristics.
3. Add a function to the "gensnd.c" file that takes as input a char type that represents the key on the phone pad above. (Letters may be entered as upper or lower case.) The function then outputs to standard out the tones that are generated by that keypress. The

sample rate of the sound is 8 kHz, and the duration of the sound is 500 ms. (milliseconds) Note that you are adding two sine waves together and you still need to keep the numbers between -1 and 1.

4. Add a function to the "gensnd.c" file that takes a sample rate parameter and duration in seconds that are float type. The function outputs to standard output sound samples that will create silence (no sound) for the specified duration at the specified sample rate.
5. Write a main program that is placed in the file "main2.c" that inputs a 10-digit telephone number from standard in and outputs to standard output DTMF tones and silence that would dial the number. The duration of the silence between tones is 250 ms.
6. Write a makefile that compiles both main files into executables. This will be the target labeled "all" in your makefile. The program in main.c produces the executable file named "gensine" and the program in main2.c produces the executable file named "gendial". Your makefile must also contain a target called "clean" that removes all object and executable files. In addition, your makefile should not be inefficient and compile or link code that is unnecessary, depending on if files have changed.
7. Approximately 10% of the points for the project will come from documentation in the files and repository.

Finally, a small hint if you read this far. The math trig functions in both C and Java use radians while all the specifications here are in hertz (cycles per second). One hertz is 2*Pi radians.