

University of Pennsylvania
ESE 112: Introduction to Electrical & Systems Engineering

Lab 9: Digital Signal Processing

Original Assignment by Bradley Zankel and Kevin Wayne
Edited by Diana Palsetia
2009-2010

Objective

- The objective for this lab is to introduce students to basics of digital signal processing
- Demonstrate use of data abstraction (object oriented programming) and array structures to solve sound synthesis problems

Introduction

In digital systems, signals are represented by a sequence of numbers or symbols. The representation and processing of these signals is known as Digital Signal Processing (DSP). DSP encompasses many subfields like: audio and speech signal processing, sonar and radar signal processing, biomedical signal processing, seismic data processing, etc. A digital computer can be used to synthesize sound, process audio signals, and compose music. In this lab, we will demonstrate fundamentals of digital audio, basic sound synthesis, and techniques for digital audio effects and processing.

Background

Sound

A music note can be characterized by its *frequency* of oscillation. For example, the music note *concert A* is a sine wave repeated 440 times per second, i.e. 440 Hertz (Hz); the note *C* is a sine wave repeated approximately 523.25 times per second. We amplify the signal to an audible level by multiplying it by a constant, say 0.8. Figure 1 shows sine wave of an A and a C with duration 15 milliseconds and amplitude 0.8. The figure for A consists of $0.015 \times 440 = 6.6$ sine waves.

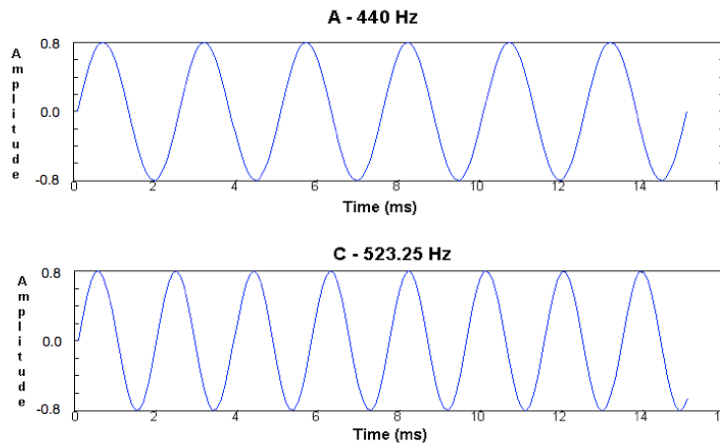


Figure 1: Notes Concert A and C

Below is a table containing the frequencies in Hertz of all twelve musical notes in the fifth octave. The ratio between the frequency of successive notes is the 12th root of two (1.05946). An *octave* is a doubling or halving of the frequency. For reference, the normal range of human hearing is between 20 and 20,000 Hz.

A	A#	B	C	C#	D	D#	E	F	F#	G	G#	A
440.00	466.16	493.88	523.25	554.37	587.33	622.25	659.26	698.46	739.99	783.99	830.61	880.00

Digital Audio

Digital audio is produced by *sampling* the instantaneous amplitude of the continuous sound wave many times a second. Each sample is a real number between -1 and +1 that represents the amplitude of the sound wave at a particular instant in time. The *sampling rate* is the number of samples taken per second. Stereophonic sound, commonly called stereo, is the reproduction of sound, using two or more independent audio channels, through a symmetrical configuration of loudspeakers, in such a way as to create a pleasant and natural impression of sound heard from various directions, as in natural hearing. Audio from CDs typically uses a sampling rate of 44,100Hz and two audio channels (left and right). For music note A, Figure 2 displays every 10th sample using a sampling rate of 44,100 and duration of approximately 1/440th of a second.

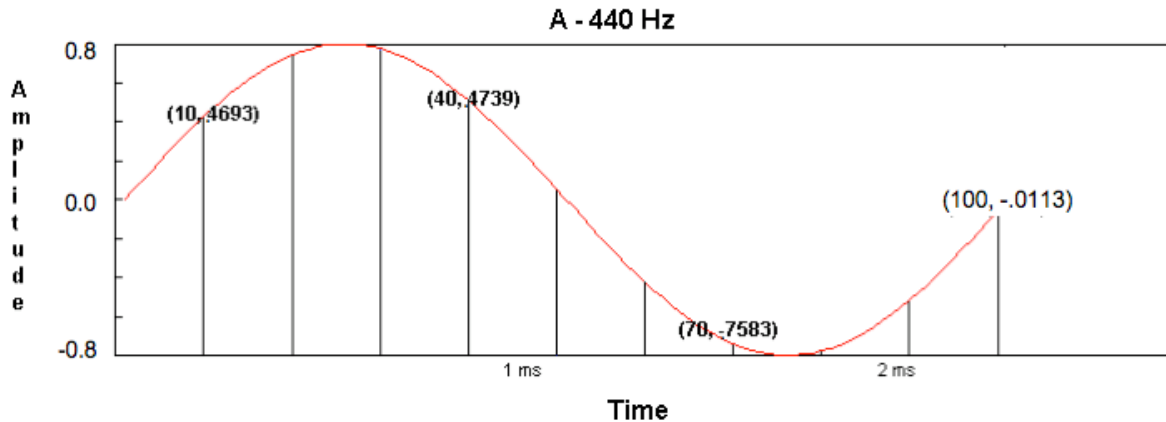


Figure 2: Every 10th Sample Music Note A

The following arrays contain the values representing the height of the wave above, at a sampling rate of 44,100:

left = { 0, .0501, .1000, .1495, .1985, .2466, .2938, .3399, .3846, .4277, .4693, ..., -.0113 }
 right = { 0, .0501, .1000, .1495, .1985, .2466, .2938, .3399, .3846, .4277, .4693, ..., -.0113 }

Sample *i* is given by $.8 \sin(2 \pi * 440 * i / 44,100)$, and we've rounded the numbers towards zero.

Drawing Canvas

Along with sound, we also will be generating visual displays found in most software music player programs such as iTunes, RealPlayer and Winamp. We provide you some already implemented classes given in java archive (jar) file. 'stdlib.jar' contains the StdDraw class that is capable of drawing shapes on a two-dimensional canvas. The canvas we are drawing on is like Quadrant I of the Cartesian plane.

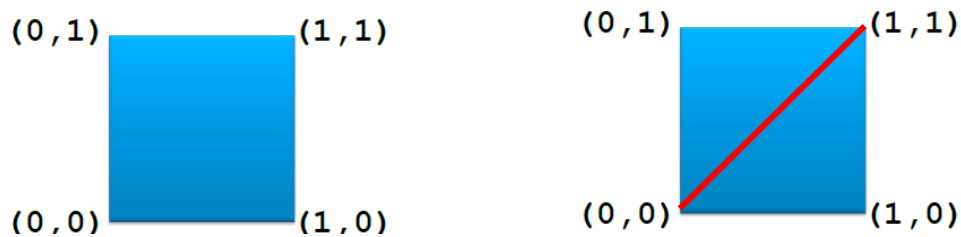


Figure 3 – Two-Dimensional Drawing Canvas

So the line method in StdDraw class can be used to draw the diagonal line (shown in red in Figure 3) as follows

```
StdDraw.line( 0, 0, 1, 1 );
```

If we want to draw a point at the center of the diagonal line we can use the point method:

```
StdDraw.point(0.5, 0.5);
```

If you want to change pen color you use setPenColor method with an input of type Color. The library defines standard colors as constants E.g. BLUE. Below is an example:

```
StdDraw.setPenColor(StdDraw.BLUE);
```

For further description and more methods see the Java documentation provided for this class:

<http://www.cs.princeton.edu/introcs/stdlib/javadoc/StdDraw.html>

Note: Most of the methods or public variables that you will use are static, i.e. if you want to use the method or variable you need to use *classname* (e.g. StdDraw.BLUE).

Java

Please refer to arrays of objects lecture notes for description and syntax

Dr Java

We will work on a regular desktop or laptop machine in this lab since the Boe-Bot cannot work with floating point numbers. In order to compile and program in part I, the IDE we will be using is Dr Java. To install Dr Java on your personal computer, follow the instructions at:

<http://www.seas.upenn.edu/~palsetia/java/installDrJava.html>

The Dr Java guide below will tell how to operate the IDE for compiling and running Java programs:

<http://www.seas.upenn.edu/~palsetia/java/drjava/primer/index.shtml>

Materials

- DrJava
- Boe-Bot with JSIDE

Pre-Lab Questions:

1. Using Dr Java

You can test this on your own computer (provided you setup Dr Java) or on the lab PCs (Start Menu > Programs > Programming Languages > Dr Java).

Create a file Test.java that contains the following code:

```
public class Test {
    public static void main(String [] args){
        System.out.println("Hello World");
    }
}
```

To compile the program, use the Compile button. To execute the program press the Run button. If everything is successful, then you see the output **Hello World** in the Interactions Pane at the bottom of the window.

2. What is sampling rate?
3. Consider the following java class:

```
public class Student {
    String name;
    int age;

    public Student(String name, int age){
        this.name = name;
        this.age = age;
    }
}
```

- a. Create an array of Student that can hold 10 Student entries. What will be the initial value of each entry in the array?
 - b. Initialize the first three entries to be three students of your choice.
4. Bring a pair of headphones to lab so you can listen to the music.

Lab Instructions

Note: As always, you will turn in all your code to BlackBoard. Make sure you comment your code well enough so that someone else can understand the progression of your program. For more information, see the Post-Lab section below.

I. Audio and Arrays on a regular PC (using Dr Java)

Your task is to write a program to generate sound waves, apply an echo filter to an MP3 file, and plot the waves.

1. Download Lab9.zip and unzip the contents. From folder PartI, add the **stdplayer.jar** and **stdlib.jar** to DrJava → Preferences → Resource Location. Click on the *Add* button to locate the .jar file. After adding both files, click *Apply*.
2. Create a Wave object template in Wave.java to store and manipulate sound wave samples so that the program **A.java** below plays concert A for 2 seconds with maximum amplitude 0.8. Similarly, **FurElise.java** plays the first nine notes of *Fur Elise*.

```
public class A {
    public static void main(String[] args){
        StdPlayer.open();
        Wave A = new Wave(440.0, 2.0, .8);
        A.play();
        StdPlayer.close();
        System.exit(0);
    }
}
```

Write the constructor:

```
public Wave(double Hz, double seconds, double amplitude)
```

It should create a new `wave` object that represents a sine wave of the specified number of Hz that is sampled 44,100 times per second over the specified number of seconds and initialize the left and right channels. The left and right channels should have the same values so that the note will play in both speakers.

Think of what data (instance variables) each Wave object should contain. Hint: you at least know that we need two arrays to represent the left and right channel

3. Next, implement the play method (method header: `public void play()`) that sends the wave data to the sound card. Use the static library method `StdPlayer.playWave(left,`

`right`), which takes as input two `double` arrays representing the left and the right channel. Note: in order to compile your code, you will need the following statement: `import javazoom.jl.player.StdPlayer;` This library is a modified version of the [JavaLayer 1.0 MP3 Player](#). (As per the GPL license, the jar file contains the original JavaLayer library and our modified source code.)

4. Now that you can play single notes, your next challenge is to play several notes at once, for example, to play a chord. To accomplish this, first add a new constructor

```
public Wave (double[] left, double[] right)
```

to `Wave.java` that takes two `double` arrays as arguments and initializes a new `Wave` object with the given data. Now, to create a chord, you can create individual notes and combine them together by writing a method:

```
public Wave plus(Wave b)
```

so that `a.plus(b)` returns the sum of `a` and `b`. To add two `Waves`, add the corresponding array entries for the left and right channels, element-by-element. Test your data type by using the program **StairwayToHeaven.java**, which is the beginning of the famous Led Zeppelin tune. Note the fifth wave played is created by the following sequence of statements:

```
Wave B6 = new Wave(493.88 * 2, .4, .4);
Wave Gs4 = new Wave(830.61 / 2, .4, .4);
Wave GsB = B6.plus(Gs4);
```

The program **MP3Player.java** decodes the MP3 file specified by the command line input and plays it. Assuming that you implemented the `Wave` API above, there is no need to write any code for this part (but you should test it and enjoy). To execute the program type the following in DrJava interactions pane:

```
java MP3Player felten.mp3
```

This program uses three new methods from the `StdPlayer` library to decode data from an MP3 file. The `String` argument to the function `StdPlayer.open()` specifies which MP3 file to use. The function `StdPlayer.getLeftChannel()` returns an array of 1,152 real numbers between -1 and +1 that are the samples of the music intended for the left speaker. The function `StdPlayer.getRightChannel()` is analogous.

5. Now that you can decode and play MP3 files, you're ready to modify the data and change the characteristics of the sound waves. An *analog filter* accomplishes this by manipulating the electrical signals that represent the sound wave; a *digital filter* does this by manipulating the digital data that represents that `Wave`. Your task is to write a program **EchoFilter.java** that implements a digital echo filter. An *echo filter* of delay 10 is a filter that adds an echo to the sound by adding the sound wave at time $t - 10$ to the

one at time t . To create this effect, maintain an array of the past 10 Wave objects and add the Wave that was originally read 10 waves ago to the current Wave. The echo filter is a client program and should be written entirely in EchoFilter.java.

For the echo filter, copy MP3Player.java to EchoFilter.java, and replace the body of main() to enable the echo effect. To test the filter, you can use **pearlharbor.mp3** which contains the speech President Roosevelt delivered after the attack on Pearl Harbor.

```
java EchoFilter felten.mp3
```

6. The final part of the assignment is to write a program **MP3Viewer.java** that takes the name of an MP3 file as a command line argument and animates both stereo channels. This program is not supposed to play the MP3 file, only to animate the sound waves. The program MP3Viewer.java should be almost identical to MP3Player.java, except that you will call draw() instead of play(). Use StdDraw.clear() to clear the screen before drawing each wave and StdDraw.show() to control the animation.

Add the following method to wave.java to plot the left channel on the top half of the screen, and the right channel on the bottom half:

```
public void draw()
```

Invoking the method draw() should plot out both channels of that wave. To implement draw() use StdDraw.point() for each of the samples. Alternatively, you could use StdDraw.line(), but the animation may be substantially slower.

It's convenient to rescale the coordinate system using StdDraw.setXscale() and StdDraw.setYscale(). Then, when you write draw(), it will be easy to scale x to match the sample index of the channel, and y to match the sample data (offset to be in the upper half for the left channel and lower half for the right channel). Feel free to use color when plotting. More methods in StdDraw class can be found in the documentation of StdDraw class. Note: it will help if you try to plot one wave before you try to plot all waves of MP3Viewer.java. So try simple file such A.java to test your draw() method.

Extra Credit (remember, there is a required Part II to the lab):

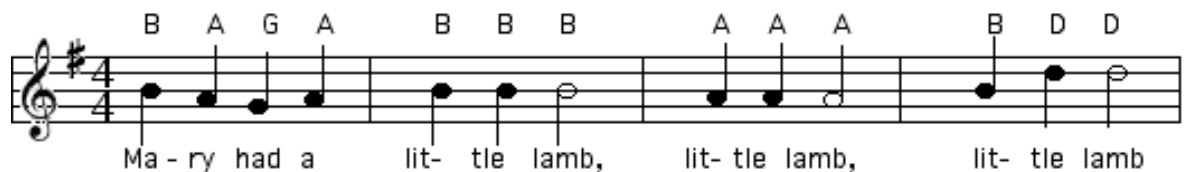
Write a program **MP3Visualizer.java** that plays the MP3 file and simultaneously displays a cool effect based on the raw data. To keep the music and animation smooth, you may need to tweak a few parameters. For example, adjust the delay in the method StdDraw.show() to keep the wave from scrolling by too fast. Also, try plotting every other wave if your computer is too slow.

II. Audio and Arrays on the Boe-Bot Platform

Because the Javelin stamp is unable to deal with floating point numbers, much of what you have done in this lab is not possible to recreate on your BoeBot. You can, however, use arrays and integers to mimic a “note player”.

The class template files are provided in lab9/PartII folder. This part has 3 tasks:

- 1) Complete the given template for `Wave.java` that has variables *frequency*, *time*, *ioPin*, and a *Freqout* class instance. The constructor for `Wave` should take in frequency, time, and the *ioPin* and create a new *Freqout* class with this information. For information on the *Freqout* class see the Javelin Stamp method reference online (in the usual Javelin Stamp javadocs on the ESE112 website). The `Wave` class should have one method called *play* that tells the *Freqout* instance to play with the specific frequency and time.
- 2) Complete the given template for `WaveSeq.java` that has as variables an array of *Wave* objects and an integer specifying the total number of notes in the array. The constructor for *WaveSeq* should take in only an array of *Wave* objects. There should be one method called *play* that loops through the array of *Waves*, playing every note sequentially.
- 3) Complete the given template for `WaveTest.java` that has only a main method. In the main method you should create *Wave* instances with different frequencies corresponding to the notes and durations of each note in the beginning of “Mary had a little lamb” shown below.



Each sequential note should be added to an array of *Waves* that will be used to create a *WaveSeq* object which you will then play. For instance, if I have quarter notes corresponding to B, A, and G called *wB*, *wA*, and *wG*, respectively, then I will want to play an array of *wB*, *wA*, *wG*, *wA* for the first measure.

For a list of notes and their corresponding frequencies, consult the website

<http://www.phy.mtu.edu/~suits/notefreqs.html>

- 4) Whatever pin you choose in `WaveTest` when creating new *Wave* objects will correspond to the pin that connects to a 5V buzzer. Connect that I/O pin to your buzzer and the other lead of your buzzer to GND.

Post-Lab Questions:

There is no official lab report for this lab. Make sure you commented all of your code so that someone else can understand the progression of your program. In particular, explain your approach to coding the following java files (include your answers as block comments at the end of your code):

Part I: EchoFilter.java, MP3Viewer.java (and MP3Visualizer.java if written)

Part II: – Wave.java, WaveSequence.java, WaveTest.java

Submit all .java files from Part I and Part II to Digital DropBox on BlackBoard. Your zipped contents should contain two folders PartI and PartII.