STMicroelectronics SensorTile : Audio Sampling and Signal Processing

Table of Contents

1. Introduction

The Lab steps provide:

1. An introduction to audio sampling from the SensorTile MEMS Microphone and the audio Analog to Digital Converter (ADC).
2. An introduction to audio output via the SensorTile audio signal Digital to Analog Converter (DAC).
3. Experience in digital signal processing. This will be provided by your modifying the AudioLoop project. Here, you will add brief C code segments that include Infinite Input Response (IIR) discrete time filters to the acquired sound signals.
4. Guidance to sample and actually hear the effects of digital signal processing.
5. Guidance to examine waveforms and spectrograms using an open source tool, Audacity.

1.1. List of Required Equipment and Materials

1) 1x STMicroelectronics SensorTile kit.
2) 1x STMicroelectronics Nucleo Board.
3) 1x Personal Computer with two USB type-A inputs OR you must have a powered USB hub.
4) 1x USB 2.0 A-Male to Micro-B Cable (micro USB cable).
5) 1x USB 2.0 A-Male to Mini-B Cable (mini USB cable).
6) 1x Headphone or Loudspeaker
7) 1x 3.5mm Audio Cable
8) 1x USB Sound Adaptor (***for Mac Users Only***)
9) Network access to the Internet.

1.2. Prerequisite T

It is recommended that users have completed and are familiar with the contents of the following tutorials before proceeding.

1. Introduction to SensorTile and the System WorkBench Integrated Development Environment (IDE)
2. Sensor System Signal Acquisition, Event Detection and Configuration.
3. Some background on digital filters (e.g. low/high pass, FIR/IIR filters)

2. Audio Sampling by AudioLoop

AudioLoop is an application included in STSW-STLKT01V1 software package, which sends audio signals acquired by the microphone to an on-board DAC via an I^2C interface. A digital MEMS microphone samples the analog sound signal and generates a Pulse-Density Modulation (PDM) stream, which is converted into a Pulse-Coded Modulation (PCM) output stream by the hardware. The output stream is then passed to the on-board DAC, allowing the user to play these sounds on speakers or headphones, or record them on a host PC. Please follow the steps below to build and launch AudioLoop on your SensorTile:

1. Open the IDE (Eclipse or System WorkBench) on your personal computer.
2. Close any project in the workspace by right-clicking “STM32L4xx-SensorTile” and selecting “delete”. Unselect the “Delete project contents on disk” box and press OK.

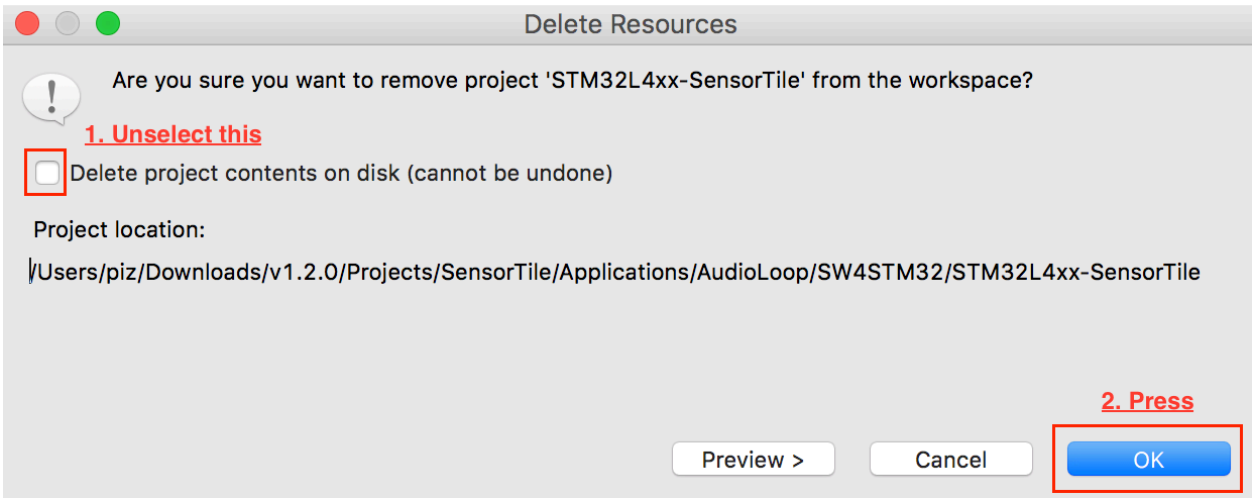


Figure 1: Deleting the existing project in your workspace.

3. Import STSW-STLKT01V1 software package, uncheck the boxes next to DataLog and BLE_SampleApp to select AudioLoop.

4. Open the file "***STM32L4XX-SensorTile > Drivers > BSP > SensorTile > SensorTile_audio_out.c***" and modify the "Includes" section by including an additional header file, as shown in Figure 2.

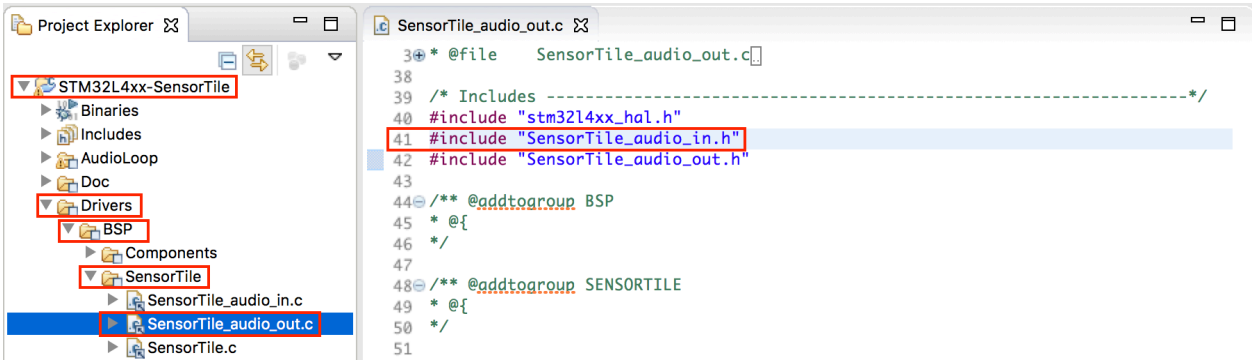


Figure 2: Including an additional header file in "SensorTile_audio_out.c".

5. Build and run AudioLoop in debug mode on the SensorTile board**.**
6. Plug a headphone or loudspeaker into the 3.5mm audio jack. If AudioLoop is running properly on your SensorTile, speak to the microphone and you will be able to hear the sound in your headphone or loudspeaker.

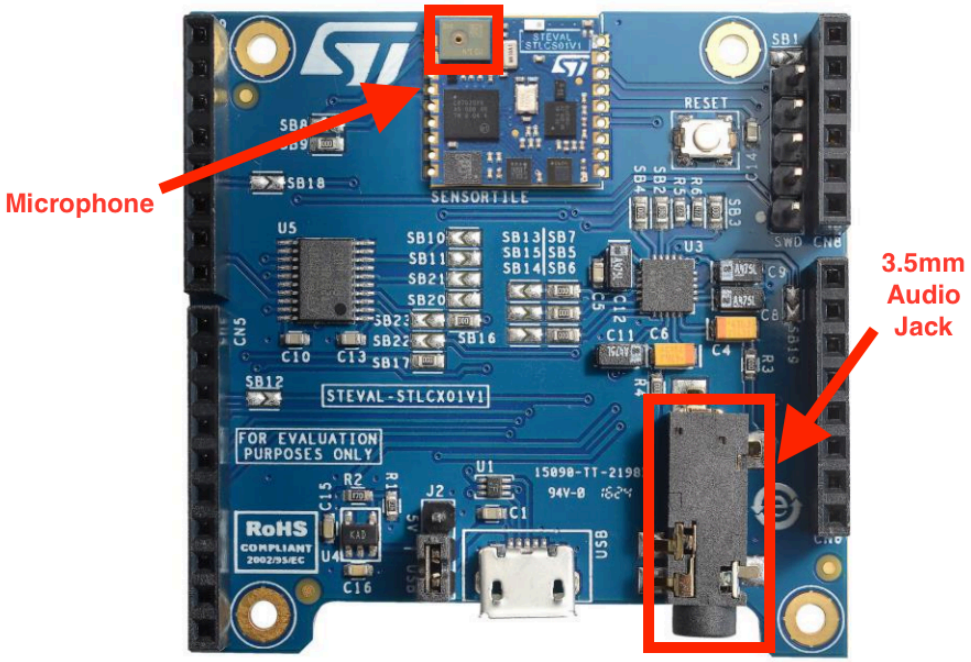


Figure 3: The locations of microphone and audio jack on SensorTile

7. Play a 4kHz test tone using your PC or Mac notebook. You can find a 4kHz test tone at the following link:

   https://www.youtube.com/watch?v=D7M7qWYFpvs

8. Play a 1kHz test tone to the microphone. You can find a 1kHz test tone at the following link:

   https://www.youtube.com/watch?v=3FBijeNg_Gs&t=27s

9. We can also generate, via speech, signals that are characteristic of human speech.
10. For most individuals, the generation of the consonant “s” letter produces a sound creating a waveform composed of signals with frequencies above 3000 Hz. To generate this sound, consider the word “hiss” with a lengthy extension of the “ss” segment.
11. Now, certain words or utterances in human speech produce low frequency sound. In particular, vowels produce low frequency sound. The vowel “o” is an ideal example. The sound generated by uttering the “boot” but not pronouncing the “b” or “t” characters and with a lengthy extension of the “oo” segment. This produces a waveform with frequencies predominantly below 1000 Hz.
12. Listen to these sounds now. We note that no filtering of the signals is applied. The sound will change in character once we apply digital signal processing filters to the audio output in the next section.

3. Digital Signal Processing – Discrete-Time IIR Filtering

3.1. Introduction

In AudioLoop, the acquired sound signals are sampled from the MEMS microphone input sensor and converted to a digital data stream by the ADC. This data is then applied to the output DAC to provide analog output signals that will be applied to the audio output. This audio output signal is then available to drive a pair of ear phones, headphones, or a speaker system.

After the data is sampled and prior to its being applied to the output DAC, we may introduce signal processing algorithms that operate on this data. This provides an outstanding introduction to signal processing and associated IoT system development.

The signal processing capability is demonstrated by adding discrete-time Infinite-Impulse-Response (IIR) filters to the original AudioLoop application in this section.

We will implement two types of digital filters: low pass filters and high pass filters. Low pass filters attenuate signals with frequencies higher than a defined corner frequency, while high pass filters attenuate signals with frequencies lower than a defined corner frequency.. Here, we will modify the AudioLoop application to add first-order low and high pass filters and examine the effects of the filters by hearing and visualizing the output signals.

3.2. Modifications in main.c

1. Open the IDE and open AudioLoop as instructed in ***Section 2: Audio Sampling by AudioLoop*** in this tutorial. Make sure that you have included the proper header file as indicated in step 4.
2. Once the IDE is opened, navigate to AudioLoop/User/main.c. Double click on main.c to open ***main.c***. Modify the “Private define” declaration section as in Figure 4.

```
/* Includes ------------------------------------------------------------------*/

#include <string.h> /* strlen */
#include <stdio.h>  /* sprintf */
#include <math.h>   /* trunc */
#include "main.h"

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
#define AUDIO_CHANNELS                      1
#define AUDIO_SAMPLING_FREQUENCY                48000
#define AUDIO_IN_BUF_LEN   (AUDIO_CHANNELS*AUDIO_SAMPLING_FREQUENCY/1000)
#define AUDIO_OUT_BUF_LEN  (AUDIO_IN_BUF_LEN*8)

/* Define for DSP -- IIR Filter start -------------*/
#define F0 3000
#define HIGH_PASS_FILTER 0
#define LOW_PASS_FILTER 1
#define NO_FILTER 2
#define GAIN 1024
/* Define for DSP -- IIR Filter end ---------------*/
```

Figure 4: Declaring DSP IIR filter parameters and variables.

3. Add the lines of code in the red box into **AudioProcess** function as shown in Figure 5.
4. Note that these define a corner frequency (3000 Hz).
5. These also define values or the conditions of Low Pass, High Pass, and No Filter operation.
6. Since we wish to perform all arithmetic with methods that avoid the requirements for floating point computation we will declare all critical variables as integer data types.
7. This leads to a limitation since some values of our signal are small integers. This leads to inevitable rounding errors in computation.
8. These rounding errors can be avoided, however, by first multiplying these variables by a gain factor, defined here as GAIN.
9. After computation, the output signal will be divided by GAIN to restore the proper amplitude.

```
123   static uint32_t IndexOut = 0;
124   static uint32_t AudioOutActive = 0;
125   uint32_t indexIn;
126
127   /* IIR filter setup */
128   int16_t filter_type = NO_FILTER;
129   int16_t IWon;
130   int16_t c[3];
131   int16_t prev_in, cur_in;
132   int32_t new_out;
133
134   IWon = (int) AUDIO_SAMPLING_FREQUENCY / (3.14159 * F0) ;
135   if (filter_type == LOW_PASS_FILTER) {
136       c[0] = GAIN / (1.0f + IWon);
137       c[1] = c[0];
138       c[2] = c[0] * (1.0f - IWon);
139   }
140   else if (filter_type == HIGH_PASS_FILTER) {
141       c[0] = GAIN - (GAIN / (1.0f + IWon));
142       c[1] = -c[0];
143       c[2] = (1.0f / (1.0f + IWon)) * (GAIN - IWon * GAIN);
144   } else if (filter_type == NO_FILTER) {
145       c[0] = GAIN;
146       c[1] = 0;
147       c[2] = 0;
148   }
149   /* IIR filter setup end */
150
151   for(indexIn=0;indexIn<AUDIO_IN_BUF_LEN;indexIn++)
```

Figure 5: Adding IIR filter setup section to the AudioProcess function.

10. Apply filters and assign filtered audio data to audio_out_buffer as shown in Figure 6.

```
150  /* IIR filter setup end */
151
152  for(indexIn=0;indexIn<AUDIO_IN_BUF_LEN;indexIn++)
153  {
154      if(indexIn == 0) {
155          audio_out_buffer[IndexOut++] = PCM_Buffer[indexIn];
156          audio_out_buffer[IndexOut++] = PCM_Buffer[indexIn];
157      } else {
158          cur_in = PCM_Buffer[indexIn];
159          new_out = cur_in * c[0] + prev_in * c[1] - audio_out_buffer[IndexOut - 2] * c[2];
160          audio_out_buffer[IndexOut++] = new_out / GAIN;
161          audio_out_buffer[IndexOut++] = new_out / GAIN;
162          prev_in = cur_in;
163      }
164
165  /*   audio_out_buffer[IndexOut++] = PCM_Buffer[indexIn];
166       audio_out_buffer[IndexOut++] = PCM_Buffer[indexIn];
167  */
168  }
169
170  if(!AudioOutActive && IndexOut==AUDIO_OUT_BUF_LEN/2)
```

Figure 6: Applying filter to audio output data.

11. Terminate and remove all previous applications from the SensorTile board.

12. Compile and run the AudioLoop application on the SensorTile board in debug mode.

13. Speak the “ss” and “oo” sounds and play the 4kHz and 1kHz test tones again. Listen to the unfiltered sounds in your headphone or speaker.

14. Change the value of the variable “***filter_type***” shown in Figure 5 from “***NO_FILTER***” to “***LOW_PASS_FILTER***” and to “***HIGH_PASS_FILTER***” and repeat step 11 to step 13.

4. Examining waveforms and spectrograms in Audacity

4.1. Introduction

This section will guide the user to examine the waveforms and spectrograms of the audio output from SensorTile using Audacity, an audio software for recording and editing. Please open the following link on a web-browser on your PC and follow the directions to download the software:

http://www.audacityteam.org/

You can examine the effect of the filters by analyzing the output waveforms and spectrograms. For more information about spectrograms, please click on the following link from Wikipedia on a web-browser on your PC:

https://en.wikipedia.org/wiki/Spectrogram

4.2. Recording and Playing a Sound in Audacity

1. Compile and run AudioLoop with filters as instructed in Section 3 of this tutorial

2. Instead of plugging in a headphone/loudspeaker, plug in one end of a 3.5mm audio cable into the audio jack of your SensorTile, and plug another end of the cable into the audio jack of your PC. Note that Mac users should have a USB sound adaptor available and connect the audio cable to the Mac through the adaptor.

3. Open Audacity on your PC, you should see a window as shown in Figure 7.

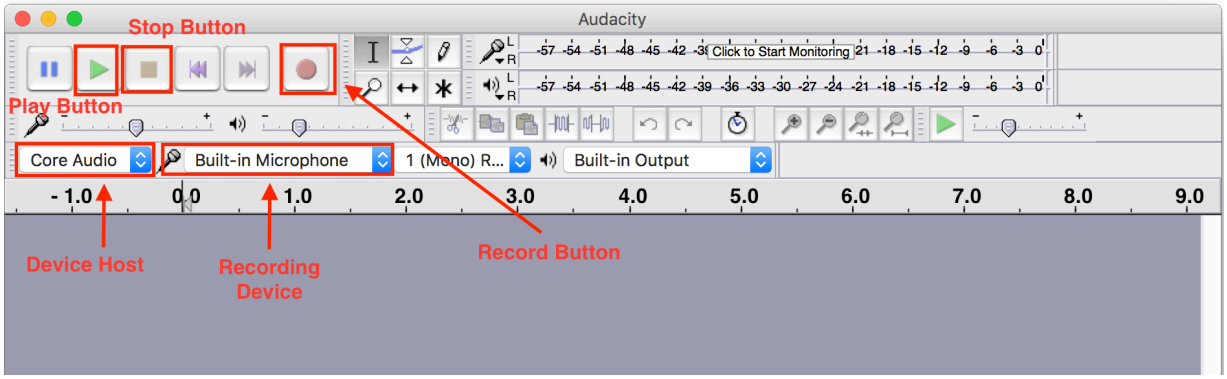


Figure 7: Audacity GUI.

For more information about the Audacity GUI, please go to ***Audacity > Help > Manual.***

4. Select the corresponding recording device to your SensorTile. In a Windows machine, the name can be “Microsoft Sound Mapper”. In a Mac machine with a USB sound adaptor connector connected, the name can be “C-Media USB Audio Device”.

5. If you cannot find the corresponding device, please restart Audacity after plugging your audio cable or your USB sound adaptor in your PC. If this solution cannot work, please consult your instructor.

6. Press the Record Button as shown in Figure 7 to start recording.

7. Press the Stop Button to stop recording. Figure 8 shows a waveform of silence (pure noise). You can close it by clicking the Close Button.

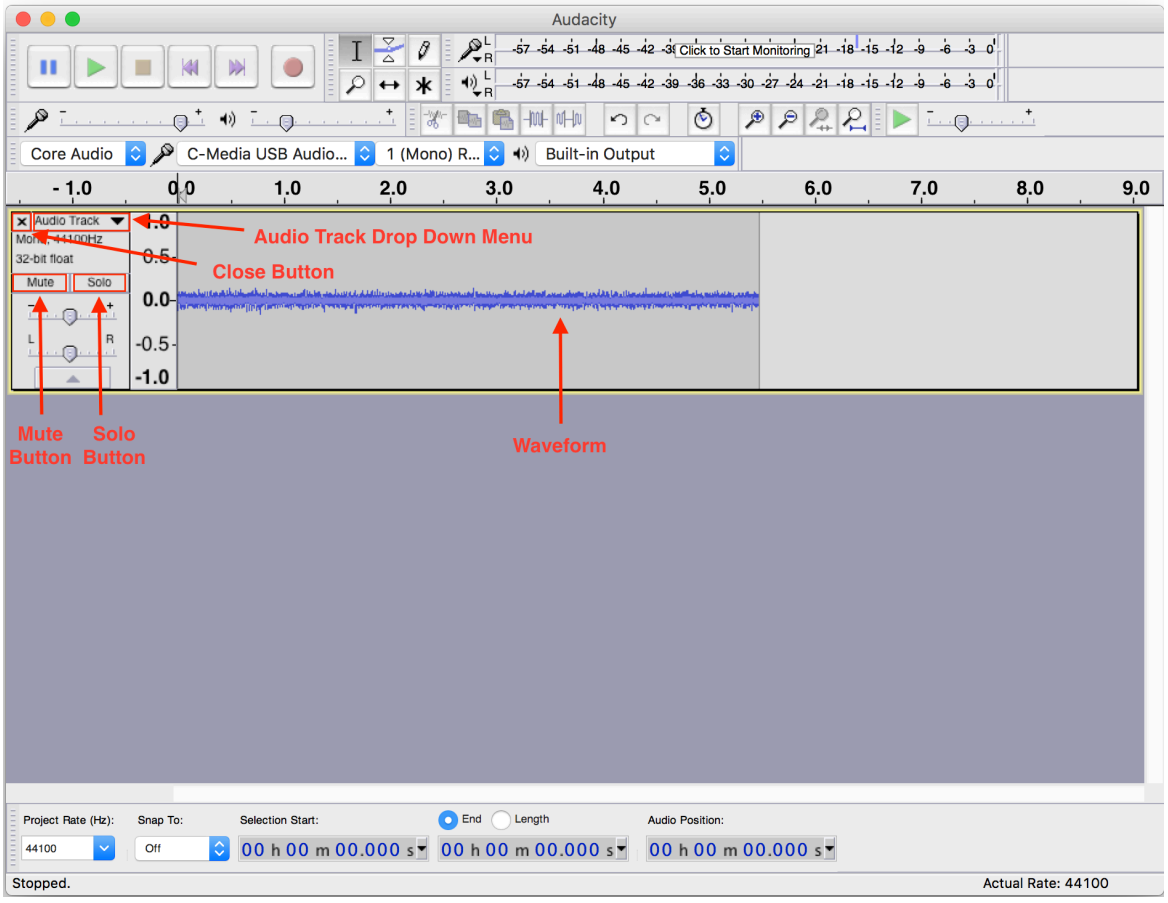


Figure 8: Audacity showing an audio waveform.

8. Press the Play Button to play the recorded sound. If there are multiple tracks available at the same time, you can click the Mute Button to mute a track, or select the Solo Button to play just that track.

4.3. Analyzing Spectrograms in Audacity

1. Click the Audio Track Drop Down Menu shown in Figure 8, select “Spectrogram”, and you should see a spectrogram as in Figure 9. Note that the blue color is the least energy and the red and white are the most.

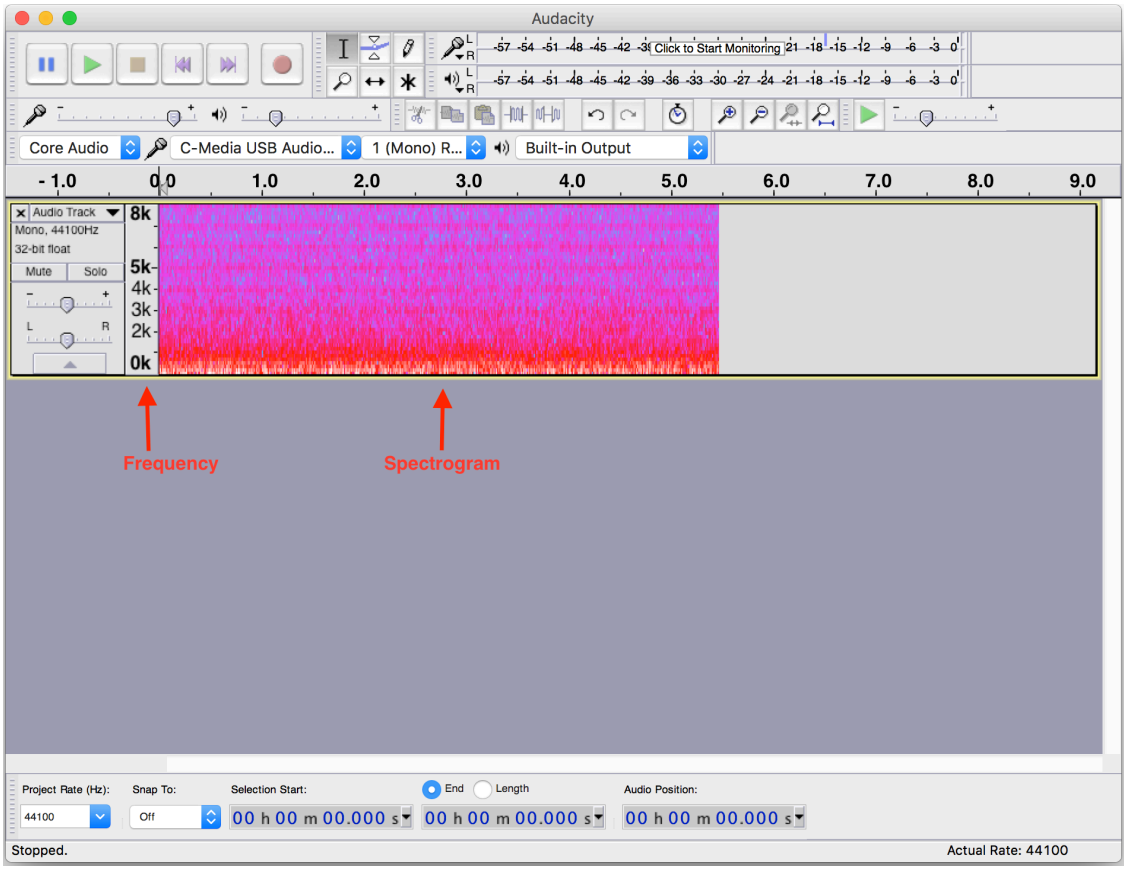


Figure 9: Audacity showing a spectrogram.

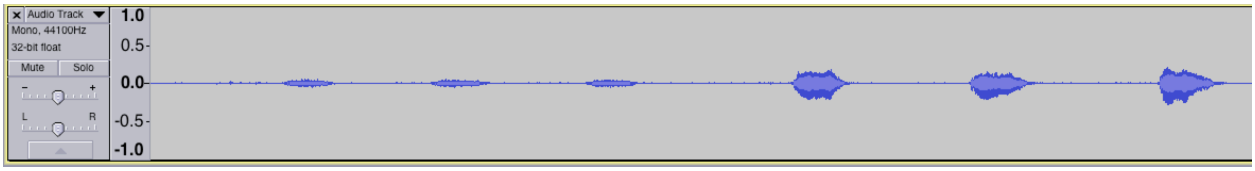
2. Examine the waveform and spectrogram of the “ss” sound after applying each type of filter (Low Pass Filter and High Pass Filter). Can you see any difference? In which frequency region does the most of energy reside for the output after each type of filter?

3. Repeat step 2 for the “oo” sound.

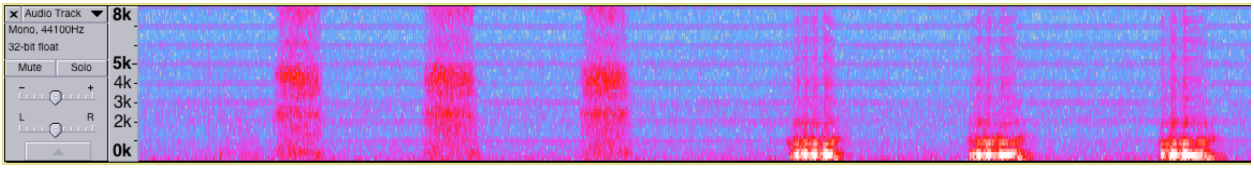
4. Repeat step 2 for the 4kHz test tone.

5. Repeat step 2 for the 1 kHz test tone.

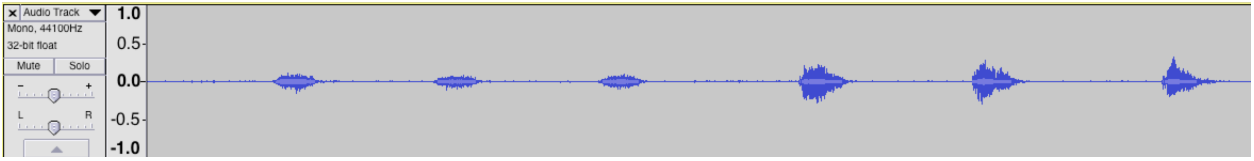
6. Please see the figures in the next page for example waveforms and spectrograms from step 2 and step 3.



Waveform display with Low Pass Filter applied with three sounds of “sss” and “ooo”. Note that the “sss” sound is attenuated in volume while the “ooo” sounds of predominantly low frequency components are preserved.



Spectrogram with Low Pass Filter applied with three sounds of “sss” and “ooo”. Note that the “sss” sound is attenuated in volume while the “ooo” sounds of predominantly low frequency components are preserved.



High Pass Filter applied with three sounds of “sss” and “ooo”. Note that the “sss” sound is increased in relative volume while the “ooo” sounds of predominantly low frequency components are now distorted.