COMP5110
Multimedia Development

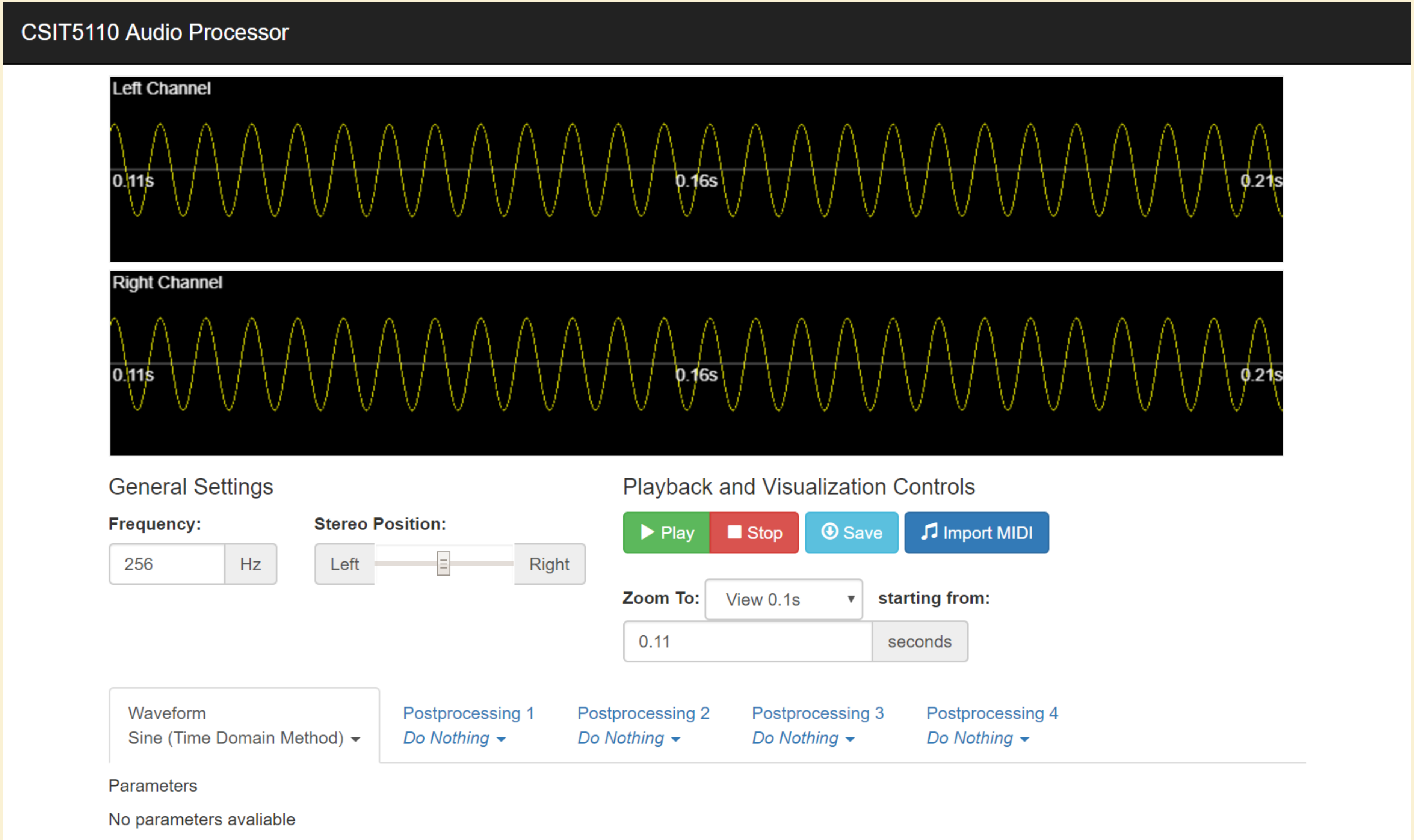
Assignment 1

Gibson Lam

Audio Processing

- This assignment covers some of the audio generation and processing techniques that we have discussed in the course
  - Additive synthesis
  - FM synthesis
  - Karplus-Strong algorithm
  - Tremolo
  - ADSR envelope
  - Pitch shift (stretching and shrinking)

The Starting System

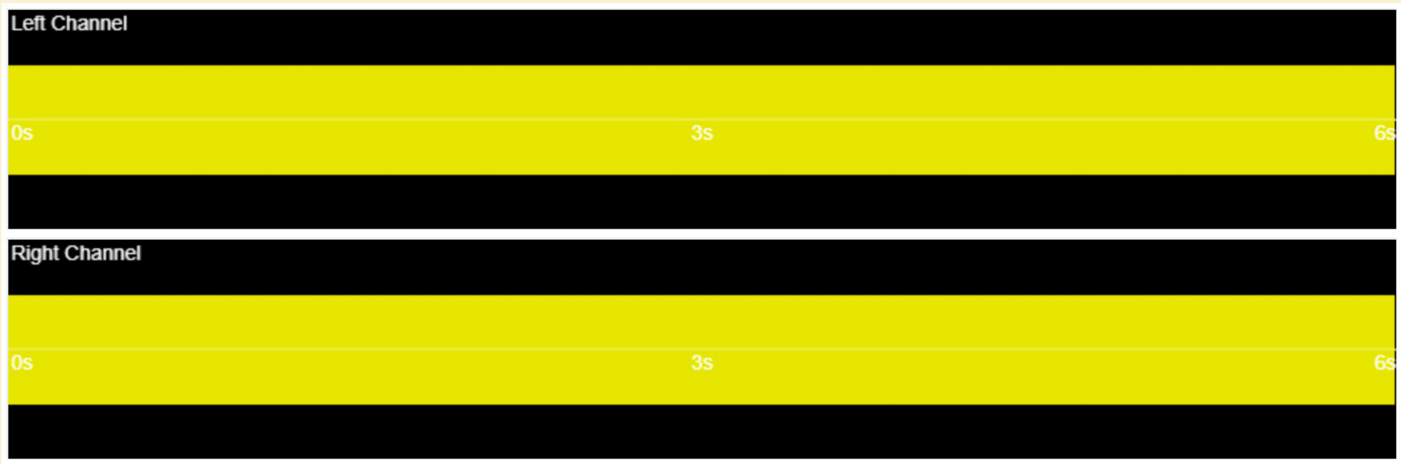


Using the Starting System

- You can download the starting system from the course canvas website
- The files of the system are put inside a zip file
- You need to extract the files from the zip file into an appropriate folder and start the system by opening `index.html` in a browser, preferably in Chrome

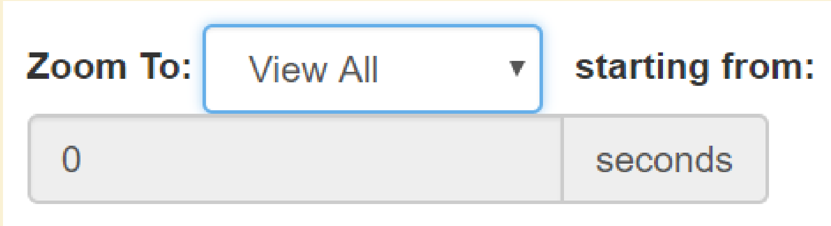
The Waveform Display

- Once you have loaded `index.html` in Chrome, you can see the waveform display at the top of the page
- It is the current sound in the system
- At the start, it is a sine wave of 256Hz

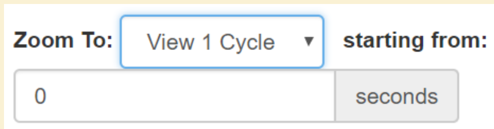


Zoom In and Out of the Display

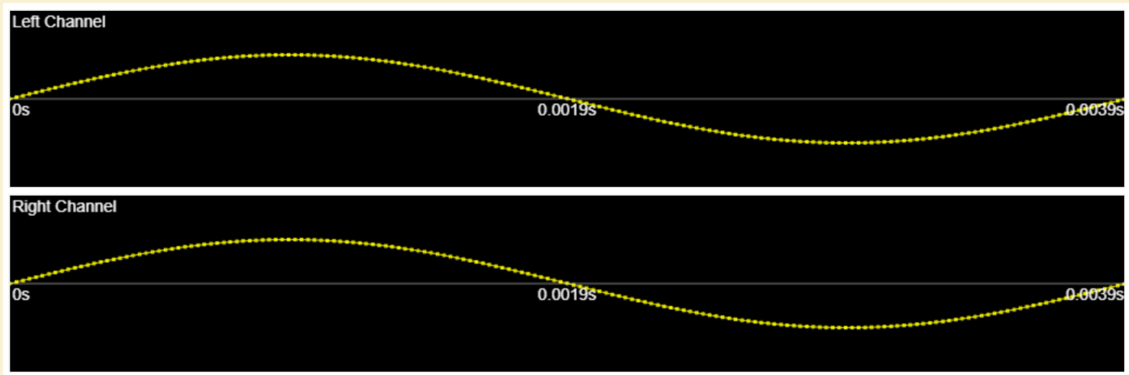
- You can zoom in and out of the waveform by using the zoom controls



- The controls are very useful if you want to examine how your waveform looks like, e.g. in one complete cycle

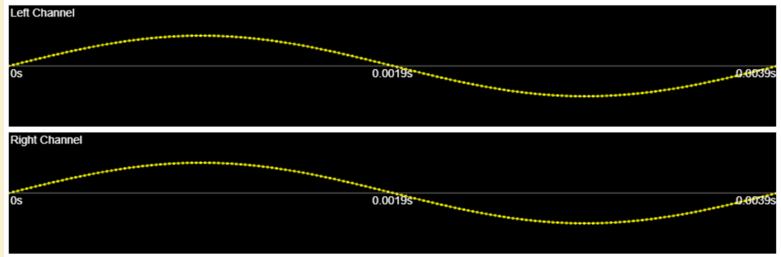


A complete cycle of the 256Hz sine wave



Mono or Stereo

Left Channel

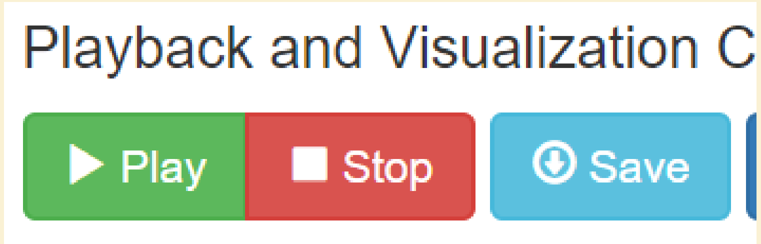


Right Channel

- As you can see from the waveform display, there are two channels, the left one and the right one
- That means when you generate the audio data you will need to generate two separate channels
- Most of the time, the system generates the same waveform in both channels so you just need to write one set of the code to work on both

Playback and Save Controls

- After generating a new sound, you can play it anytime by using the playback controls



- You can also save the sound into a wav file using the Save button

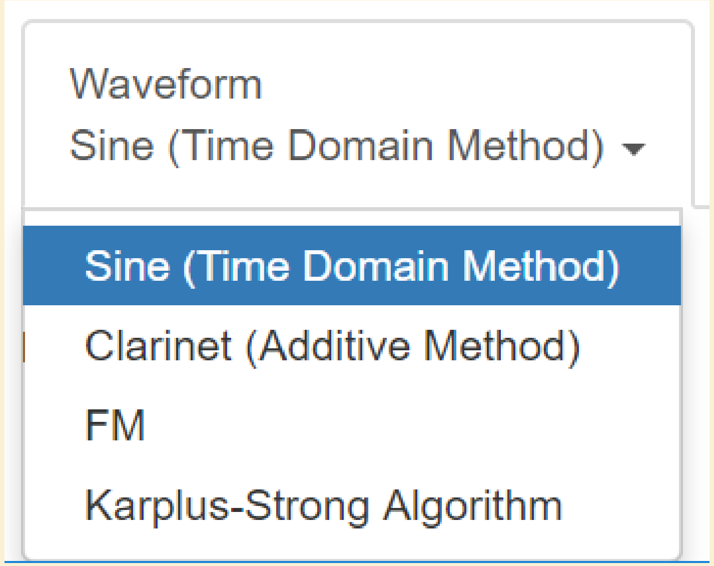
General Settings

- The general settings allow you to change the frequency and the stereo positioning of the generated sound

- Note that the duration of the generated sound is 6 seconds long and therefore you can see 6 cycles in the display if the frequency is 1Hz

Audio Generators

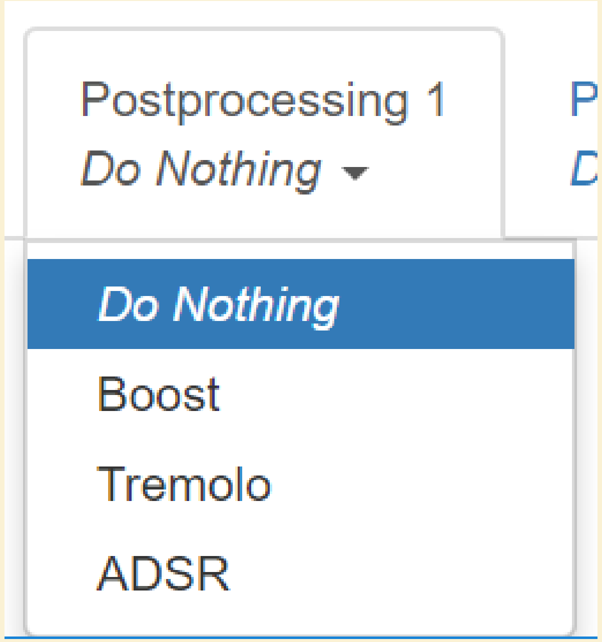
- One major function of the system is to generate sounds
- If you click on the `Waveform` selection box, you will see the list of available sound generators



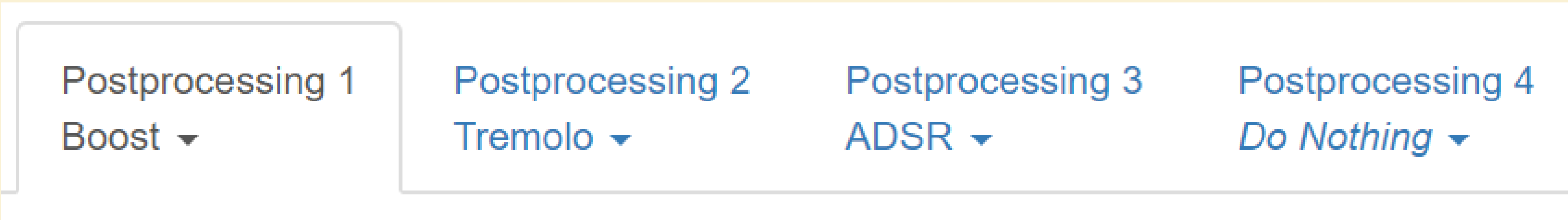
- The sine wave generator has been given, you will need to complete the rest

Audio Post-processors

- Similarly, if you click on any of the Postprocessing selection box, you will see the list of available post-processors
- Like the sine wave generator, boost has been given to you
- You will need to finish the two remaining post-processors



Multiple Post-processors



- If you look at the tabs, you will see there are four post-processors that can be selected
- The system allows you to use at most four of them, one after another

Importing Songs

- The last functionality of the system is to import a MIDI song
- Once you click on the `Import MIDI` button, you can select a MIDI song from the given music folder in the system

Using the Import MIDI Window

- You first choose your MIDI file, adjust some options and then generate the music
- You can see the pitch shift option, which is what you will need to complete later on

File Structure

`index.html`

Starting HTML file

`js`

- audioController.js
- audioPlayback.js
- audioSequence.js
- binaryToolkit.js
- channel.js
- main.js
- pitchShift.js
- postprocessor.js
- utility.js
- waveformGenerator.js
- waveTrack.js

JavaScript files, all your code will be added here

`music`

- art_of_fugue.json
- beauty_and_the_beast.json
- beauty_and_the_beast_intro.json
- let_it_go.json
- let_it_go_short.json
- moonlight_sonata.json
- moonlight_sonata_intro.json
- phantom_of_the_opera.json
- phantom_of_the_opera_intro.json
- small_world.json
- small_world_intro.json
- turkish_march.json
- turkish_march_intro.json

Music files, for the import MIDI function

Sine Wave Generation

- The sine wave generator has been given in the starting system
- You can see the code from the JavaScript file, `waveformGenerator.js`, as shown below:

```
case "sine-time": // Sine wave, time domain
    for (var i = 0; i < totalSamples; ++i) {
        var currentTime = i / sampleRate;
        result.push(amp *
                    Math.sin(2.0 * Math.PI *
                    frequency * currentTime));
    }
    break;
```

Studying the Code

- The generators that you will work on will use very similar code
- There are some given variables in the code:

`frequency`	the generated frequency
`nyquistFrequency`	the Nyquist frequency
`totalSamples`	the total number of samples
`sampleRate`	the sample rate
`amp`	the maximum amplitude of the waveform
`result`	the array storing the samples which is empty initially

- You will likely use a for loop to keep pushing (appending) samples to the `result` array

Additive Synthesis – Clarinet

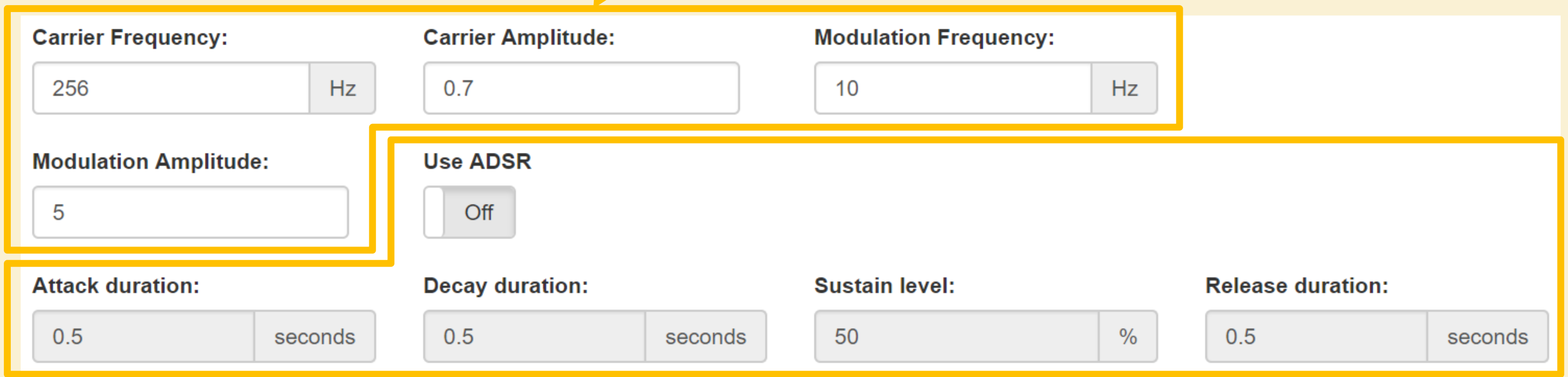
- You will create a clarinet sound generator using additive synthesis
- For the harmonics you can find them in the additive synthesis notes
- For this generator, you should not generate anything bigger than or equal to the Nyquist frequency

FM Synthesis

- You will create FM sounds in the assignment
- Inputs to the FM sounds include the frequencies and amplitudes of the carrier and modulator
- In addition, you can optionally apply an ADSR envelope to the modulator amplitude

FM Synthesis Parameters 1/2

Basic FM Parameters



Optional ADSR envelope for the modulation amplitude

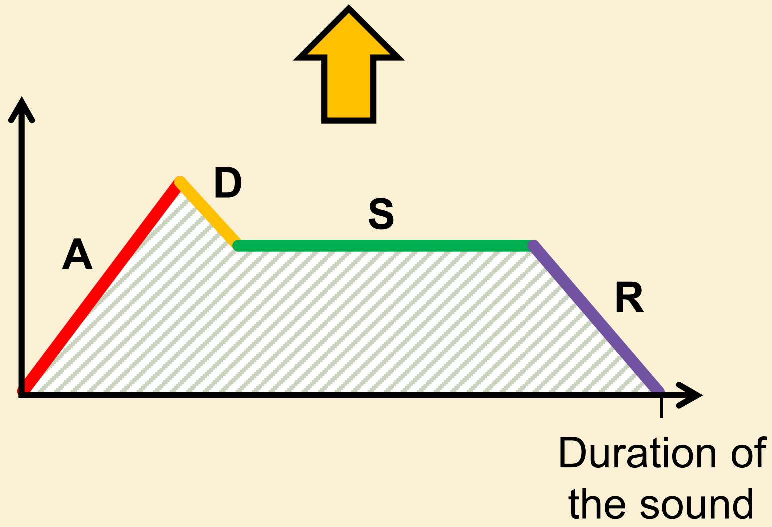
FM Synthesis Parameters 2/2

- Basic FM parameters
  - These include the frequencies and amplitudes of the carrier and modulator
- ADSR parameters
  - The four parameters define the ADSR envelope: attack time, decay time, sustain level as a percentage and the release time
  - Note that when a sound is generated with a particular duration, the release time is included within the duration of the sound

ADSR Envelope

- The ADSR envelope can be optionally applied to the modulator amplitude, i.e.:

$$a_c * \sin(2 * PI * f_c * t + \boxed{a_m} * \sin(2 * PI * f_m * t))$$



Making the Envelope

- To apply the envelope in the code, you first calculate the multiplier according to the current time within the envelope
  - A helper function `lerp()` has been created for you so that you can find the interpolation between two values
- The multiplier can then be multiplied to the modulator sine wave in the FM formula

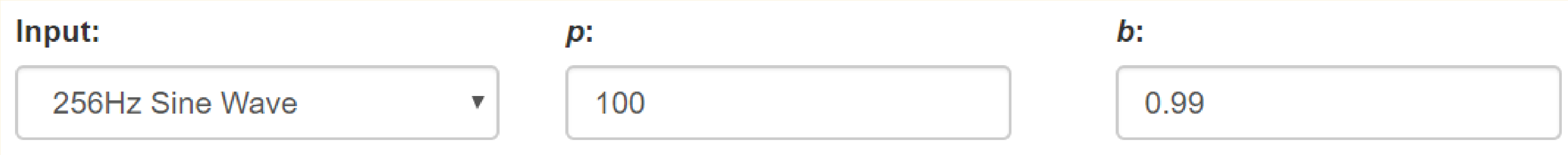
The lerp() function

- The `lerp()` function takes three parameters: `value1`, `value2`, `percentage`
- Based on the value of `percentage`, the function will return a value between `value1` and `value2`
- For example, `lerp(0,5,0.75)` returns the value `3.75`
- You will likely use the `lerp()` function for the attack, decay and release sections in the envelope

Karplus-Strong Algorithm

- You will create a Karplus-Strong generator
- You will use the extended Karplus-Strong algorithm, i.e. you will include the blend factor in the generation of the sound
- There are a few parameters that you can adjust before generating the sound, as shown in the next slide

Karplus-Strong Parameters



- Input
  - The initial energy that can be a sine wave or simple white noise
- p
  - The size of the delay line
- b
  - The blend factor

The Initial Energy

- The given code has already generated the initial energy for you
- The energy is either a 256Hz sine wave or white noise filling up the entire duration of the sound buffer
- That means when you work on the code, you will replace the samples in the `result` array rather than ‘push’ new samples in

Boost

- Boost has been given to you in the starting system
- You can read its code, shown on the next slide, in the file `postprocessor.js`
- The main loop of the code reads the audio data of each channel and then modifies the data array inside the channel
- You will use similar code for the two post-processors you have to complete

The Boost Loop

- Here is the boost loop in `postprocessor.js`:

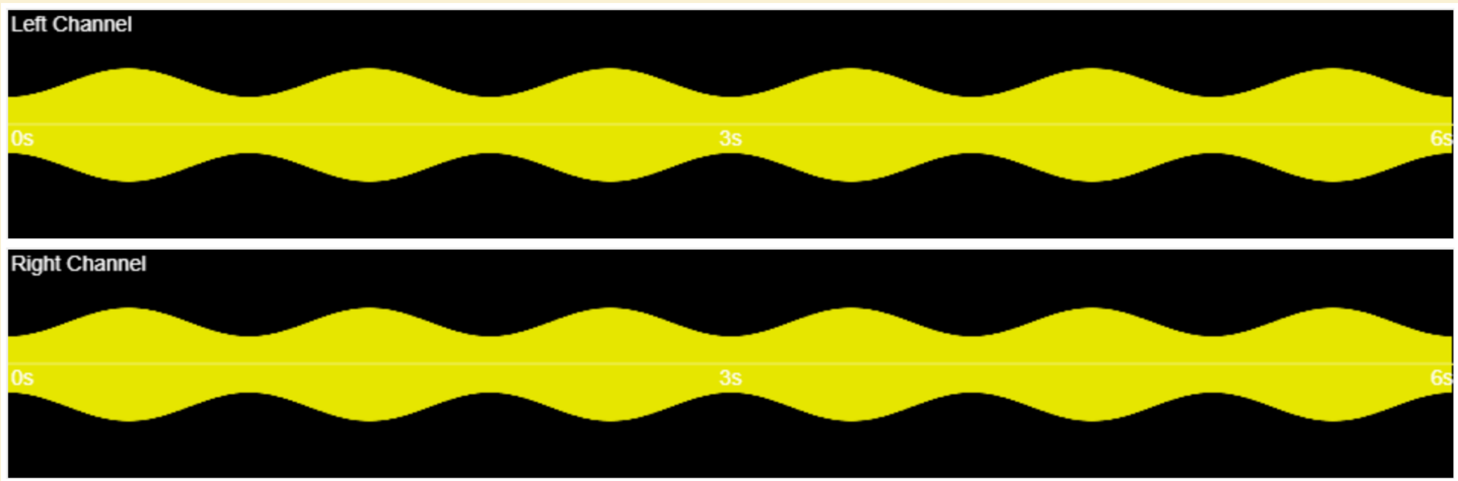
```
// Post-process every channels
for(var c = 0; c < channels.length; ++c) {
    // Get the sample data of the channel
    var audioSequence = channels[c].audioSequenceReference;

    // For every sample, apply a boost multiplier
    for(var i = 0; i < audioSequence.data.length; ++i) {
        audioSequence.data[i] *= multiplier;
    }

    // Update the sample data with the post-processed data
    channels[c].setAudioSequence(audioSequence);
}
```

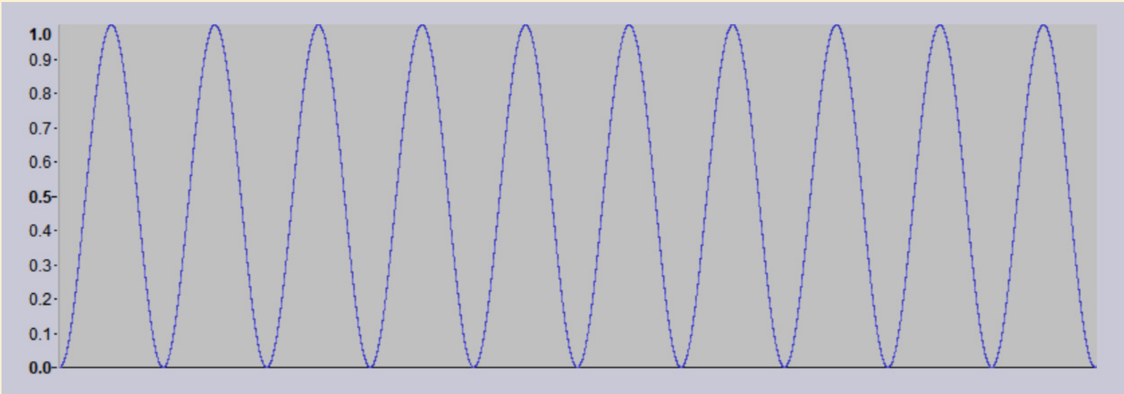
Tremolo

- You can apply tremolo to your sound
- Here is what you see if you apply a 1Hz tremolo with 0.5 wetness to a 256Hz sine wave



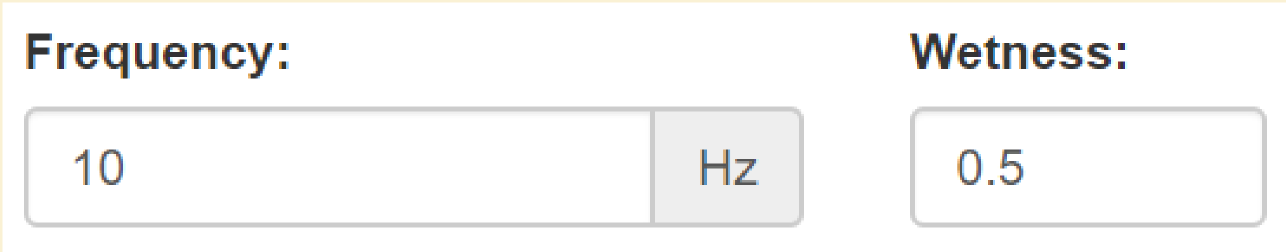
The Multiplier

- You need to make sure that the tremolo multiplier starts from a value of 0, i.e. the sine function is shifted to start in an appropriate position



- The wetness value then controls how ‘high’ is the bottom part of the multiplier

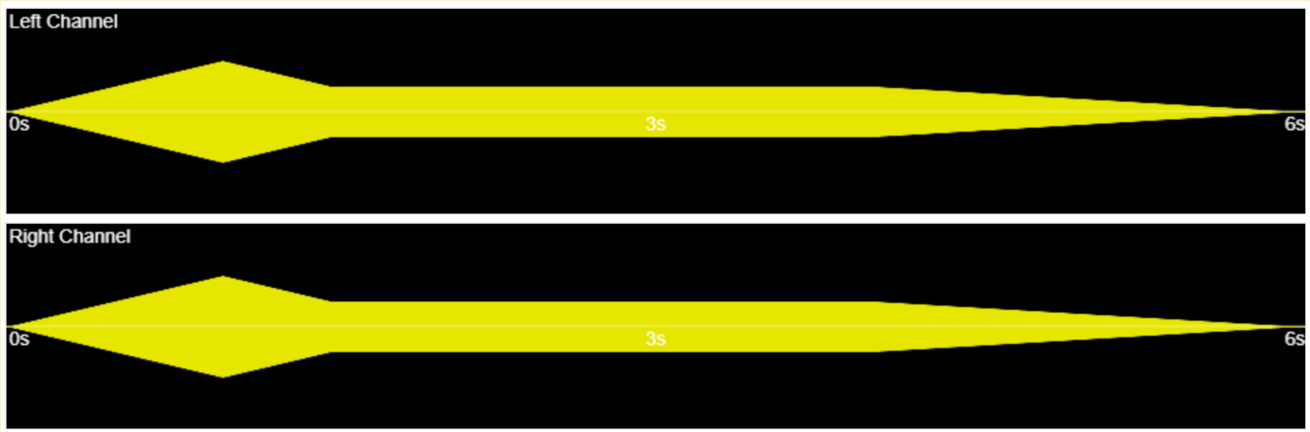
The Tremolo Parameters



- Frequency
  - The frequency of the tremolo multiplier
- Wetness (0 – 1)
  - The wetness of the multiplier

ADSR Envelope

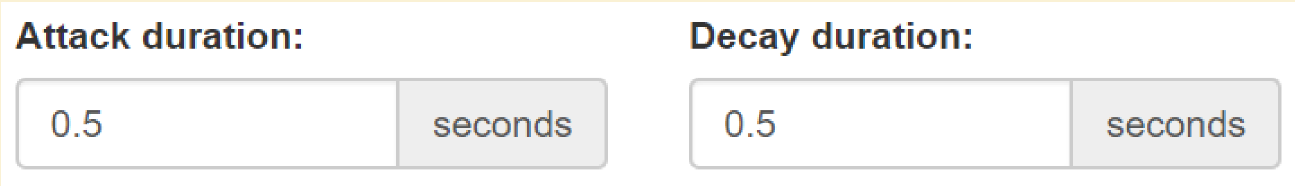
- This is the same ADSR envelope that you have implemented in the FM generator
- In this part, the ADSR envelope can be applied to the final waveform, like this:

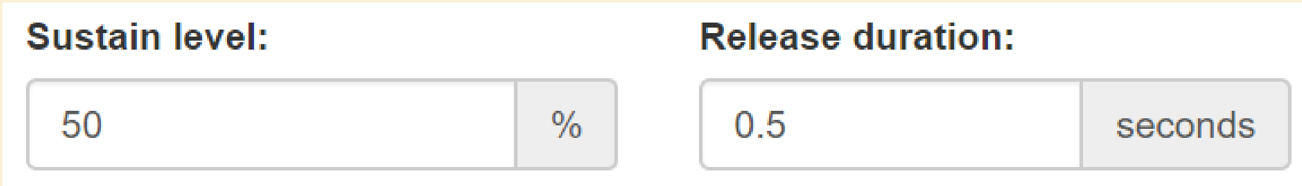


Here the ADSR envelope has been applied to the 256Hz sine wave, with the following ADSR parameters:

A = 1s, D = 0.5s, S = 50% and R = 2s

The ADSR Parameters

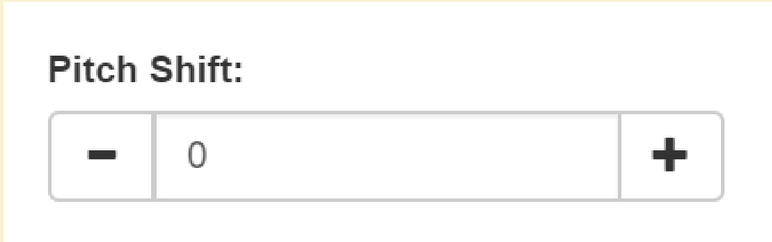




- These parameters define the ADSR envelope as before
- Their values have been read into some variables in the given code

Pitch Shifting

- In the Import MIDI window, you can select the amount of pitch shifting for the MIDI song



- Although you can directly change the MIDI pitches when the audio data is generated, you are required to do it in a slightly harder way, i.e. changing the pitch in the digital audio level

Doing Pitch Shifting

- You will do the pitch shifting code in the file `pitchShift.js`
- First, you need to calculate the frequency ratio based on the pitch shift value
- The ratio then determines the way samples are skipped or duplicated in the resulting audio
- Because you are doing stretching and shrinking, the final duration of the sound will not be the same as the original song

Marking Scheme 1/2

- Total marks is 100
  - Additive synthesis
    - A clarinet sound can be generated with the correct harmonics — 5%
    - No harmonics can go over the Nyquist frequency — 5%
  - FM synthesis
    - Basic FM sounds can be generated by adjusting the frequencies and amplitudes of the carrier and modulator — 10%
    - Appropriate ADSR envelope can be applied to the modulator amplitude — 15%
  - Karplus-Strong algorithm
    - Basic Karplus-Strong algorithm works correctly (b = 1) — 5%
    - Drum-like sounds can be produced when b = 0.5 — 10%

Marking Scheme 2/2

- Total marks is 100
  - Tremolo
    - Tremolo with the correct frequency is applied 5%
    - Tremolo with the correct phase is applied 5%
    - Tremolo with the correct wetness is applied 5%
  - ADSR envelope
    - Appropriate ADSR envelope can be applied to the amplitude of the sound over the entire duration 15%
  - Pitch shift (stretching and shrinking)
    - The MIDI song can be correctly stretched for a reduced pitch 10%
    - The MIDI song can be correctly shrunk for an increased pitch 10%

Submission

- The deadline of the assignment is:

8pm, Saturday, 27 Oct 2018

- To submit your assignment:
  - You need to put **everything** (HTML file, JavaScript files and song files) into a zip file called `<your ITSC account>_a1.zip`
  - For example, if your ITSC account is *johnc*, you will put your files into `johnc_a1.zip`
  - You can then submit the zip file through canvas