Equaliser Coursework Submission — CM2208

Kieran Williams

I. Q1.

Overview of Basic Implementation:

To implement the basic functionality of the multi band equaliser I utilised algorithms taken from the DAFX: Digital Audio Effects Book. The algorithms I used include a shelving filter [1], and a peaking/ notching filter [2] both implemented by the code author Jeff Tackett. I implemented a Shelving filter for the low pass filter with a frequency cutoff at 200Hz and high pass filter with a 12KHz frequency cutoff. For all intermediary filters I used a peaking/ notching algorithm with the ranges specified by a frequency centre and frequency bandwidth to produce the following ranges: 200-400 Hz, 400-800 Hz, 800-1200 Hz, 1.2-1.8 KHz, 1.8-3 KHz, 3-5 KHz,

The GUI displays both the original and the filtered waveform in two separate axes. The master volume is then used to multiply the filtered signal's amplitude to increase or decease it.

Overview of Novel Feature:

I have also made use of the wah wah example from David Marshall's Digital Audio FX slides [3]. The wah can be turned off or on by selecting or unselecting the wah-wah button. I have provided the user with control over the frequency of the wahs (discrete values: 250 500, 750, 1000, 2000, 4000, 6000 Hz), the min cutoff frequency (250, 300, 400, 500) and the max cutoff frequency (2, 3, 4, 5 KHz), and the dampener through using a slider with values from 0.02 to 0.4. The user is also able to save the filtered audio as an audio file.

Non-Filtering Controls (File select, preset select, play, play original, pause, stop, reset sliders, save filter preset, save audio)

The user is able to import an audio file by clicking the select file button and choosing a file in one of the following audio formats: Wav, mp3, aac, flac, m4a, aiff

This audio is then loaded into the application along with its audio waveform plotted onto the 'original audio' axes, the calculateFilters function is then also run and the output which should be the same as the original is plotted on the 'filtered audio' axes. These test audio files can be found in the audio_samples directory provided. The samples and sample frequency for both the original and filtered audio are stored globally in the mlapp file for access for different uses such as recalculating the filter, playing the original audio.

The filter slide values can be stored as .dat file by clicking the save preset button. These values can then be loaded into the application by clicking the select preset button in the top right. Upon loading in a preset, the slider values will be set to the preset files respective values and audio filters will be calculated and displayed.

Kieran Williams (e-mail: williamskm8@cardiff.ac.uk)

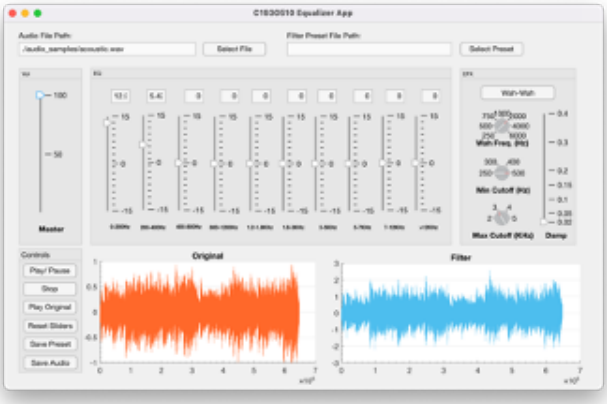

Fig. 1. Equaliser app with increased volume and bass boost

When Play/ pause is pressed, if the globally saved audioplayer object is not already playing it is passed in as an argument to the resume function. If it is currently playing the audio will be passed into the pause function.

When Stop is pressed, the globally saved audioplayer object is passed in as an argument to the stop function.

When the reset filters button is pressed, all of the slider values are returned to their default values and the function calculateFilters is called

Upon the Save Audio button being pressed, the user is able to input a a file name in a dialogue and save the audio using the audiowrite function with the current audioplayer object passed in as an argument.

Project Structure and Implementation:

All functionality is written in .m files located in the functions directory. This folder is added to the path of the .mlapp file in the project's root directory. The mlapp callbacks execute the functions that should get run upon an event such as a button click, slider changed, etc... These functions are typically passed the app object by reference so that the function's have access to the global public variables (samples & sampling rate of both the filtered audio and the original audio). The current audioplayer is also a public attribute of the app object to allow for different functions to make use of the currently in use audioplayer. All UI components are publicly accessible in the app object, allowing me to update any UI values/ component changes within the separated out .m functions. By separating out the functions into their own files, it allows for the codebase to be far more modular and easier to work with.

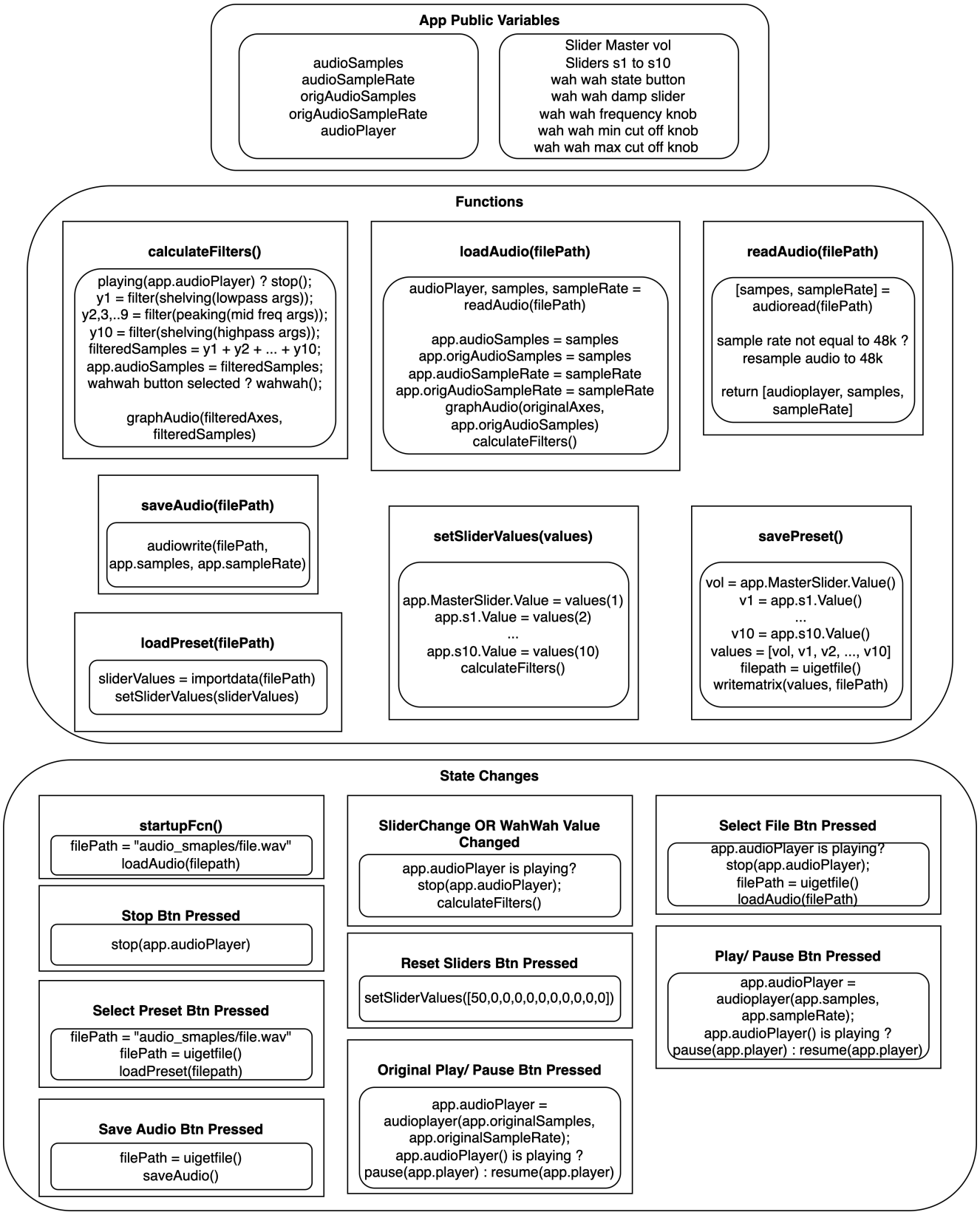

Fig. 2. Code Structure and Implementation Overview Diagram

Algorithm explanations

Shelving Filters

The shelving filters boost or cut high or low frequencies via a cutoff frequency and gain. The shelving Algorithm works by checking the type parameter for "Base_Shelf" or "Treble_Shelf" throwing an error if type is anything else. It then progresses to achieve a second order shelving filter (second order to increase the slope of the shelving filter). The values K is calculated as stipulated in The DAFX book [4]:

K = tan(πfc/f) and V0 = 10G/20

These values are used in the following equations for filtering either high end frequencies or low end frequencies.

If the V0 value calculated is less than 1 then it is reinstantiated as the reciprocal of its previous value. This is done to handle negative gain values (the gain sign is used to indicate the boost or cut).

If gain is > 0, and the type argument of the shelving function is "Base_Shelf" then a bass boost is possible by calculating the following cut-off frequency parameters which can be calculated as follows:

```
b0 = (1 + sqrt(V0)*root2*K + V0*K^2) / (1 + root2*K + K^2);
b1 =             (2 * (V0*K^2 - 1) ) / (1 + root2*K + K^2);
b2 = (1 - sqrt(V0)*root2*K + V0*K^2) / (1 + root2*K + K^2);
a1 =                (2 * (K^2 - 1) ) / (1 + root2*K + K^2);
a2 =             (1 - root2*K + K^2) / (1 + root2*K + K^2);
```

Fig. 1.

[1]

If gain is < 0, and the type argument of the shelving function is "Base_Shelf" then a bass cut is possible by calculating the following cut-off frequency parameters which can be calculated as follows:

```
b0 =            (1 + root2*K + K^2) / (1 + root2*sqrt(V0)*K + V0*K^2);
b1 =               (2 * (K^2 - 1) ) / (1 + root2*sqrt(V0)*K + V0*K^2);
b2 =            (1 - root2*K + K^2) / (1 + root2*sqrt(V0)*K + V0*K^2);
a1 =            (2 * (V0*K^2 - 1) ) / (1 + root2*sqrt(V0)*K + V0*K^2);
a2 = (1 - root2*sqrt(V0)*K + V0*K^2) / (1 + root2*sqrt(V0)*K + V0*K^2);
```

Fig. 2.

[1]

If gain is > 0, and the type argument of the shelving function is "Treble_Shelf" then a treble boost is possible by calculating the following cut-off frequency parameters which can be calculated as follows:

[1]

If gain is < 0, and the type argument of the shelving function is "Treble_Shelf" then a treble cut is possible by calculating the following cut-off frequency parameters which can be calculated as follows:

```
b0 = (V0 + root2*sqrt(V0)*K + K^2) / (1 + root2*K + K^2);
b1 =             (2 * (K^2 - V0) ) / (1 + root2*K + K^2);
b2 = (V0 - root2*sqrt(V0)*K + K^2) / (1 + root2*K + K^2);
a1 =               (2 * (K^2 - 1) ) / (1 + root2*K + K^2);
a2 =            (1 - root2*K + K^2) / (1 + root2*K + K^2);
```

Fig. 3.

```
b0 =              (1 + root2*K + K^2) / (V0 + root2*sqrt(V0)*K + K^2);
b1 =                 (2 * (K^2 - 1) ) / (V0 + root2*sqrt(V0)*K + K^2);
b2 =              (1 - root2*K + K^2) / (V0 + root2*sqrt(V0)*K + K^2);
a1 =            (2 * ((K^2)/V0 - 1) ) / (1 + root2/sqrt(V0)*K + (K^2)/V0);
a2 = (1 - root2/sqrt(V0)*K + (K^2)/V0) / (1 + root2/sqrt(V0)*K + (K^2)/V0);
```

Fig. 4.

[1]

If gain is 0 then an all pass filter is performed returning the values as 0s with b0 being the V0 value.

The returned values are the cut-off frequency parameters which can be used with the filter function to perform the already discussed filters. These return are:

```
a = [  1, a1, a2];
b = [ b0, b1, b2];
```

Fig. 5.

[1]

Peaking Filters

The peaking filter boosts or cuts mid frequency bands with a cutoff frequency, a bandwidth and a gain value. The values K and V0 are calculated as follows:

K = tan (πfc /fs)

V0 = 10^(G/20); [1]

These values are used in the following equations for filtering either high end frequencies or low end frequencies.

If the V0 value calculated is less than 1 then it is reinstantiated as the reciprocal of its previous value. This is done to handle negative gain values (the gain sign is used to indicate the boost or cut).

If gain is > 0, Then a boost is made possible by calculating the cut-off frequency parameters:

```
b0 = (1 + ((V0/Q)*K) + K^2) / (1 + ((1/Q)*K) + K^2);
b1 =         (2 * (K^2 - 1)) / (1 + ((1/Q)*K) + K^2);
b2 = (1 - ((V0/Q)*K) + K^2) / (1 + ((1/Q)*K) + K^2);
a1 = b1;
a2 =  (1 - ((1/Q)*K) + K^2) / (1 + ((1/Q)*K) + K^2);
```

Fig. 6.

[2]

If gain is < 0, Then a cut is made possible by calculating the cut-off frequency parameters:

```
b0 = (1 + ((1/Q)*K) + K^2) / (1 + ((V0/Q)*K) + K^2);
b1 =       (2 * (K^2 - 1)) / (1 + ((V0/Q)*K) + K^2);
b2 = (1 - ((1/Q)*K) + K^2) / (1 + ((V0/Q)*K) + K^2);
a1 = b1;
a2 = (1 - ((V0/Q)*K) + K^2) / (1 + ((V0/Q)*K) + K^2);
```

Fig. 7.

[2]

The returned values are the cut-off frequency parameters which can be used with the filter function to perform the already discussed filters. These return are:

```
a = [  1, a1, a2];
b = [ b0, b1, b2];
```

Fig. 8.

[2]

Wah Wah

To perform the wah wah effect, the algorithm follows a state variable filter. Tuning coefficients F1 and Q1 are calculated as:

F1 = 2sin(πfc/fs), and Q1 = 2d [3]

Where F1 is the difference equation coefficients and Q1 dictates the size of the passbands (how aggressive each wah should be). Delta is calculated by Fw/ Fs which gives the change in centre frequency per sample.

A triangle wave of centre frequency values is created by

```
Fc=minf:delta:maxf;
while(length(Fc) < length(x) )
    Fc= [ Fc (maxf:-delta:minf) ];
    Fc= [ Fc (minf:delta:maxf) ];
end
```

Fig. 9.

[3]

The triangle wave is clipped to the length of the audio samples.

F1 and Q1 are then calculated as stated above.

The difference equations are then calculated as such:

[3]

This iterates over every sample and calculating the difference equations every time Fc changes due to the n value increasing. yb is the filtered samples that are built up with each subsequent iteration. The samples a re normalised and set to the app objects public filtered samples.

[3]

[1], [2], [3], [4]

REFERENCES

[1] J. Tackett, "Shelving Filters." [Online]. Available: https://uk.mathworks.com/matlabcentral/fileexchange/16568-bass-treble-shelving-filter?s_tid=prof_contriblnk


```
yh=zeros(size(x));
yb=zeros(size(x));
yl=zeros(size(x));
yh(1) = x(1);
yb(1) = F1*yh(1);
yl(1) = F1*yb(1);
for n=2:length(x)
    yh(n) = x(n) - yl(n-1) - Q1*yb(n-1);
    yb(n) = F1*yh(n) + yb(n-1);
    yl(n) = F1*yb(n) + yl(n-1);
    F1 = 2*sin((pi*Fc(n))/Fs);
end
```

Fig. 10.

```
maxyb = max(abs(yb));
yb = yb/maxyb;
app.y = yb;
```

Fig. 11.

[2] ——, "Peaking Filters." [Online]. Available: https://uk.mathworks.com/matlabcentral/fileexchange/16567-peaking-notch-iir-filter?s_tid=prof_contriblnk

[3] D. Marshall, "Audio FX." [Online]. Available: http://www.cs.cf.ac.uk/Dave/CM0268/PDF/10_CM0268_Audio_FX.pdf

[4] "DAFX," in *DAFX: Digital Audio Effects (Second Edition)*, U. Zolzer, Ed.