Assignment 3 - Audio System Controller
Real Time DSP
ECS732P Queen Mary University

David Moffat - 130670191
d.j.moffat@se13.qmul.ac.uk

April 26, 2014

Abstract

An audio system controller is an audio engineering industry tool that is used for setting up and controlling a PA system. These principally have multiple input, multiple output setup and a series of digital audio effects and routing options to allow the user complete control of their audio system. This paper constructs a basic, one in two out audio processor with crossover, delay, compression and parametric equalisation control, with all software being run on a Beagle Bone Black with Audio Cape.

1 Introduction

The purpose of this paper is to describe and justify the implementation of a real world loudspeaker system controller, built on a Beagle Bone Black (BBB) with audio cape. Audio system controllers or are used through out the professional audio engineering industry, where a controller with multiple input and multiple output will allow a user to tune a sound system. Generally these system controller will have a range of crossovers, parametric equalisation filters, graphic equalisation filters, phase control, gain control, delay buffers and compressor limiters all user controllable with variable sample rate and routing control. This paper presents an audio system controller that has been and designed and built an originally written specification.
This paper will go on to discuss the dap and hardware design and control concepts in Section 2. The physical hardware of the system will then be explained in Section 3 and the software implementation is described in Section 4. Further to this the audio system controller will then be evaluated in Section 5 and conclusions drawn about this paper, including further work, presented in Section 6.

2 Design

This Section will describe the design of the audio system controller. Based on the original specification as presented in Section 2.1, this will be reviewed and a series of specific design decisions will be presented in section 2.2.

2.1 Specification

The original specification document [5] stated that the following elements should ideally be presented as part of this paper:

Crossover Hardware control of crossover frequency and filter order

Parametric Eq Hardware control of frequency, quality factor and gain (Minimum 1 per output)

Compressor/Limiter Hardware control of threshold, ratio, attack and release per output

Delay Hardware control of delay per output

Noise Gate Hardware control of threshold, attack and release.

Through the development of this project, a series of the original specification points were modified as explained in Section 2.2

2.2 Design Decisions

A series of design decision were made as part of this project, and they are summarised in this section.

- A simple Linkwitz Riley second order crossover was implemented with , no filter order control.
- Two parametric equalisers were implemented per output.

- One compressor limiter was implemented for the input only.
- Delay lines of 0s to 0.5s which is the same as a delay of 0m to 170m.
- A fractional delay was implemented to reduce audio clipping and noise while the delay was being changed.
- No noise gates were implemented.
- Hardware controls were implemented in a paging manor as discussed in Section 3.
- A basic compressor with no knee function was implemented.
- All implementation was in floating point C, no assembly or fixed point was used.

2.2.1 Crossover Design

A linkwitz riley second order crossover was used and the coefficients are calculated from following equations.

Linkwitz Riley 2nd Order Low Pass Filter

$$\theta_c = 2\pi f_c / f_s \tag{1a}$$

$$\Omega = \pi f_c \tag{1b}$$

$$\kappa = \frac{\Omega_c}{\tan(\theta_c} \tag{1c}$$

$$\delta = \kappa^2 + \omega_c^2 + 2\kappa\Omega_c \tag{1d}$$

$$a_1 = \frac{-2\kappa^2 + 2\Omega_c^2}{\delta} \tag{1e}$$

$$a_2 = \frac{-2\kappa\Omega_c + \kappa^2 + \Omega_c^2}{\delta} \tag{1f}$$

$$b_0 = \frac{\Omega_c^2}{\delta} \tag{1g}$$

$$b_1 = 2\frac{\Omega_c^2}{\delta} \tag{1h}$$

$$b_2 = \frac{\Omega_c^2}{\delta} \tag{1i}$$

Linkwitz Riley 2nd Order High Pass Filter

$$\theta_c = 2\pi f_c / f_s \tag{2a}$$

$$\Omega = \pi f_c \tag{2b}$$

$$\kappa = \frac{\Omega_c}{\tan(\theta_c} \tag{2c}$$

$$\delta = \kappa^2 + \omega_c^2 + 2\kappa\Omega_c \tag{2d}$$

$$a_1 = \frac{-2\kappa^2 + 2\Omega_c^2}{\delta} \tag{2e}$$

$$a_2 = \frac{-2\kappa\Omega_c + \kappa^2 + \Omega_c^2}{\delta} \tag{2f}$$

$$b_0 = \frac{\kappa^2}{\delta} \tag{2g}$$

$$b_1 = -2\frac{\kappa^2}{\delta} \tag{2h}$$

$$b_2 = \frac{\kappa^2}{\delta} \tag{2i}$$

2.2.2 Compressor Design

The compressor works on a simple peak detector. The input signal in translated in to a dB level, and then if the peak audio level is detected, The region above the threshold is scaled by 1/ the compression ratio. Fundamentally the compressor works as such:

$$O = \begin{cases} I & \text{if} \quad I < T \\ T + (T - I) \times \frac{1}{R} & \text{if} \quad I > T \end{cases}$$

Where O is the output value in dB, I is the input sample in dB, T is the threshold value in dB and R is the compression ratio.
Further to this, some ballistics are implemented to smooth the values based on the attack and release times.

$$O = \begin{cases} I_{n-1}\alpha_a + (1 - \alpha_a)\delta_I & \text{if} \quad I > I_{n-1} \\ I_{n-1}\alpha_r + (1 - \alpha_r)\delta_I & \text{if} \quad I \leq I_{n-1} \end{cases}$$

Where I_{n-1} is the previous input sample, α_a is the attack value and α_r is the release value calculated by

$$\alpha_x = \mathrm{e}\left(\frac{-1}{f_s x_t}\right)$$

where x represents either attack or release and x_t is the attack or release time in seconds.
The output of this function O is then used as a gain factor for the original input.

2.2.3 Parametric Equaliser Design

Four parametric filters were implemented. Each pair of filters were cascaded, providing two filters per output. The second order peaking filters were implemented using the following equations. Due to the nature of peaking filters, there was also a wet and dry mix coefficients, and as such the original audio input had to be used in combination with the output of the filter.

$$\theta_c = 2\pi f_c / f_s \tag{3a}$$

$$\mu = 10^{Gain(dB)/20} \tag{3b}$$

$$\zeta = \frac{4}{1+\mu} \tag{3c}$$

$$\beta = 0.5\frac{1-\zeta\tan(\frac{\theta_c}{2Q})}{1+\zeta\tan(\frac{\theta_c}{2Q})} \tag{3d}$$

$$\gamma = (0.5+\beta)\cos\theta_c \tag{3e}$$

$$a_1 = -2\gamma \tag{3f}$$

$$a_2 = 2\beta \tag{3g}$$

$$b_0 = 0.5-\beta \tag{3h}$$

$$b_1 = 0.0 \tag{3i}$$

$$b_2 = -(0.5-\beta) \tag{3j}$$

$$\text{'Wet Mix'} = \mu - 1 \tag{3k}$$

$$\text{'Dry Mix'} = 1 \tag{3l}$$

3 Hardware

Within this section, the hardware layout is presented and explained. Figure 1 shows the circuit diagram of the primary hardware flow diagram. This circuit presents six potentiometers and six LED's, where each potentiometer can be used an individual value control. The LEDs are presented as wight he use of circuit 2, the six analog controls have been paged, allowing four pages of six analog controls. As such a total of twenty four analog controls are presented with this control system.

The paging of control values is a simple concept, however some minor details must be noted. To ensure that values do not jump around significantly while changing pages, there is a full user interface developed, utilising the LEDs and switched. If the control page is changed, through changing one of the switches, all values from that page are stored fixed. On the new page, all potentiometers are set as having no hardware control, and all the LEDs are turned off. Then, if the value of the potentiometer is within a small window range of the stored value for that page, the potentiometer is activated and the LED turned on. As such, there should never be a significant value jump on any of the control values, as the potentiometer should be within 10mV range from the previously saved value.

The hardware was then connected up to the P9 header on the BBB, as per table 1

Each one of the potentiometer, with a specific switch combination, translate into a specific hardware control of a value as can be seen in table 2

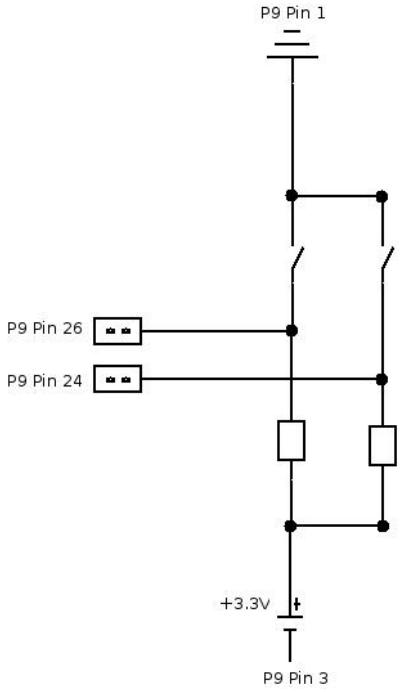

Figure 2: Analog Control Page Circuit Diagram

Table 1: Hardware Connection Control Pins

Hardware Element	Header	Hardware Pin	Software Connection
Switch 1	P9	26	GPIO 14
Switch 2	P9	24	GPIO 15
LED 1	P9	13	GPIO 31
LED 2	P9	11	GPIO 30
LED 3	P9	15	GPIO 48
LED 4	P9	17	GPIO 5
LED 5	P9	21	GPIO 3
LED 6	P9	22	GPIO 2
Potentiometer 1	P9	39	Analog 0
Potentiometer 2	P9	40	Analog 1
Potentiometer 3	P9	37	Analog 2
Potentiometer 4	P9	38	Analog 3
Potentiometer 5	P9	33	Analog 4
Potentiometer 6	P9	36	Analog 5

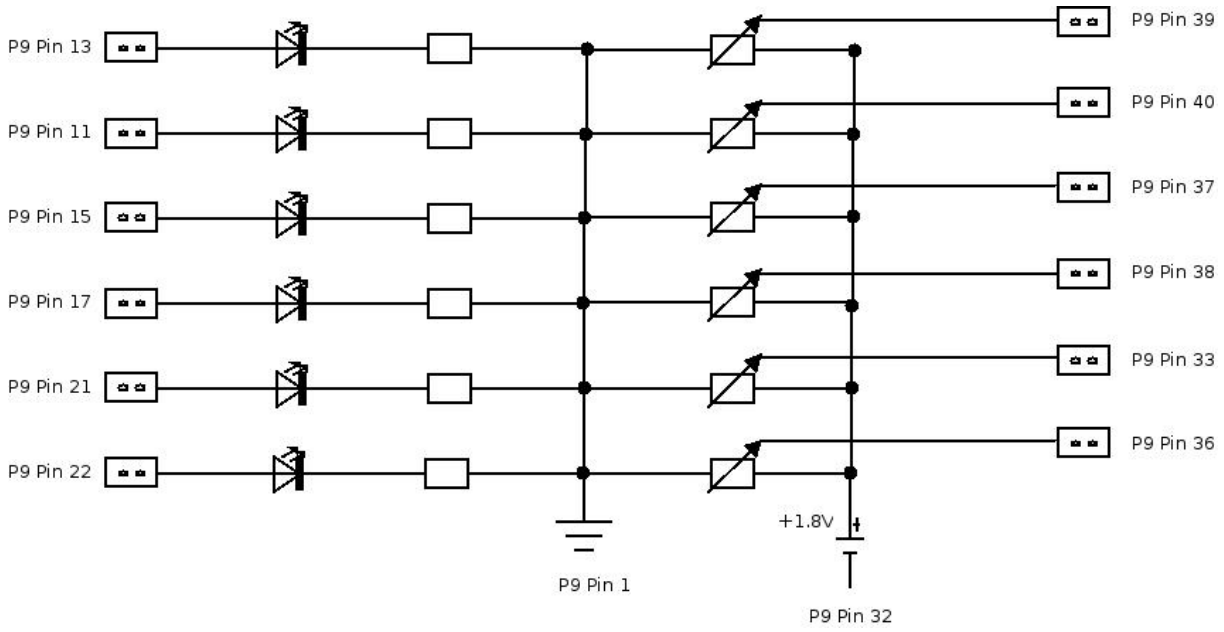

Figure 1: Analog Control Circuit Diagram

Table 2: Hardware Control Matrix

Overall Control Switch Setting	Crossover and Delay Off Off	Compressor Off On	Left Parametric Eq On Off	Right Parametric Eq On On
Potentiometer 1	Crossover Frequency	Threshold	Frequency 1	Frequency 1
Potentiometer 2	Left Channel Gain	Ratio	Quality Factor 1	Quality Factor 1
Potentiometer 3	Left Channel Delay	Unused	Gain 1	Gain 1
Potentiometer 4	Right Channel Gain	Attack	Frequency 2	Frequency 2
Potentiometer 5	Right Channel Delay	Decay	Quality Factor 2	Quality Factor 2
Potentiometer 6	Unused	Make-up Gain	Gain 2	Gain 2

4 Software

This section provides a high level description and explain of the software produced for this paper. A multithreaded approach is taken for the software in this program. There are threads to this program, the hardware thread, which will be explained in Section 4.1 and the audio thread, which will be explained in Section 4.2.

4.1 Hardware Thread

The purpose of the hardware thread is to read all hardware inputs and perform any non time-critial calculations. As such, recalculating coefficients for filters are all part of the hardware thread, despite the impact they have on the audio, they are primarily driven by the hardware and only required to be calculated then the hardware changes a control value. This section will go on to provide a summary of every function within the hardware thread.

4.1.1 cmpfunc

This function was taken from tutorialspoint.com [10]

- Comparison function of two integer values

4.1.2 median

- Sort array data into numeral order, using cmpfunc
- return centre value in array

4.1.3 readAnalogPin

- Read an analog pin a predefined number of times
- Apply a median filter to the results
- Return result

4.1.4 updateControlValues

- Read selected analog pin, given pin number and global variable gButtonFlag
- Normalise analog results to required range
- Store result in global variable gControlValues

4.1.5 sensorLoop

- Initialise required variables and define hardware pin connections as per Table 1
- Export and set pin direction for GPIO, both for buttons and LED's
- Initialise analog pin values to reasonable start values
- Begin main hardware loop, continuous loop until interrupt command sent
  - Check if button value had been changed and store in buttonStatus variable, using bit shifting
  - If buttonStatus has changed reset analog control using gAnalogControl variable, using bit shifting
  - Loop over all analog pins
    - If current potentiometer has control of value and the value has changed, update analog control value and call both updateControlValues and updateHardwareValues functions
    - If the current potentiometer does not had control but is within a threshold, allow control of the analog value to be taken by the potentiometer, update the gAnalogControl flag, update the current value and call updateControlValues and updateHardwareValues functions
  - Set LED status based on gAnalogControl, so if a potentiometer has control of a value, turn the LED on.
  - Sleep for 1ms, to allow other threads to proceed, due to low priority of the hardware thread
- Unexport all GPIO pins
- Exit thread

4.1.6 updateHardwareControl

- If hardware value updated requires filter coefficients to be updated, update relevant filter coefficients
- This ensures that filter coefficients are only updated when required, for efficiency

4.1.7 calcCrossCoef

- Read in control values
- If the crossover frequency is less than 1Hz, Lock thread, set coefficients as a pass through and unlock thread
- Otherwise perform calculations for filter coefficients
- Lock thread to ensure coefficients cannot be changed simultaneously to applying coefficients

- Update crossover filter coefficients in global variables
- Unlock thread

4.1.8 calcParaCoef

- Given a specific filter index, read in required hardware values,
- If the filter gain is 0dB, Lock thread, set coefficients as a pass through and unlock thread item Otherwise perform calculations for filter coefficients
- Lock thread to ensure coefficients cannot be changed simultaneously to applying coefficients
- Update parametric eq filter coefficients in global variable
- Unlock thread

4.2 Audio Thread

The audio thread is created with high priority. This is the time critical thread, where all processing must take place is the most efficient manor possible. As such significant effort was put into making the audio thread more efficient and trying to move as much code as possible to the hardware thread.
This section will go on to provide a high level explanation of each function within the audio thread.

4.2.1 initialise

- Memory allocation for all required variables
- Initialise Mutex's
- Initialise all delay buffer lines
- Initialise all filter coefficients

4.2.2 render

- Sum audio channels together to produce a mono input track
- Calculate compression control value
- Apply crossover to mono track and send each channel to an output
- Calculate and apply fractional delay to each channel
- Apply parametric equalisation to each channel
- Apply compression and gain to each output channel
- Store outputs to audio buffer

4.2.3 setDelayReadPointer

- Calculate each delay read pointer value based on current delay, ensuring smooth transitions between current delay input delay setting

4.2.4 compressor

Based on code from Reiss and McPherson [9]

- Given in input audio signal, read hardware control values
- Calculate compression gain value
- Store compressor gain value in global variable

4.2.5 applyCrossCoef

- Lock thread to ensure crossover coefficients cannot be changed by hardware thread while calculating the audio outputs
- Read in global variables for both low pass and high pass filter coefficients
- Calculate each channel output
- Unlock thread to allow threading to continue
- Store previous inputs and outputs and update pointers

4.2.6 applyParaCoef

- Loop through all parametric equalisers
  - Lock thread to ensure parametric equalisation coefficients cannot be changed by hardware thread while calculating the audio outputs
  - Read in global variables for both filter coefficients
  - Calculate filter output
  - Unlock thread to allow threading to continue
  - Store previous inputs and outputs and update pointers

4.2.7 cleanup

- Free all memory allocation

5 Evaluation

A series of visualisations were produced by means of proving that various aspects of this audio system controller works in an effective manor. This section will provide some visualisations of example aspects of the system and evaluate the overall performance. Conclusions will then be drawn and discussed in Section 6

A sin sweep was played though the BBB audio system controller set as a generic pass through and the outputs were plotted in both the time-frequency domain in Figure 3 and in the time domain, in Figure 4. These figures will be used as a comparison basis for further plots within this paper.

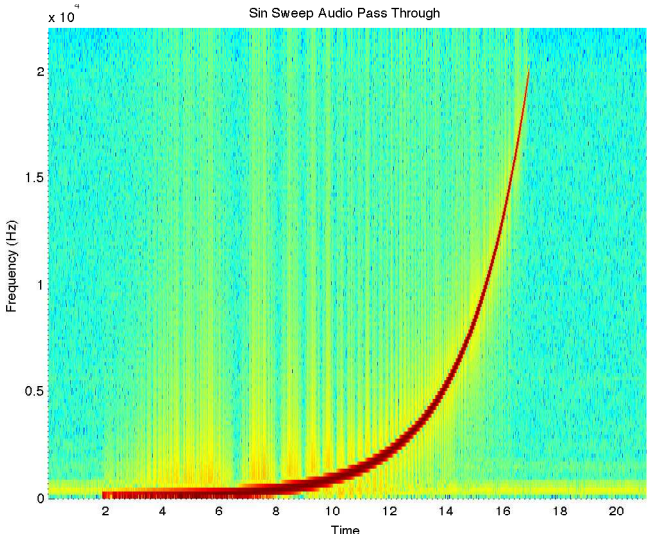

Figure 3: Plot of Audio Pass Through Spectrogram

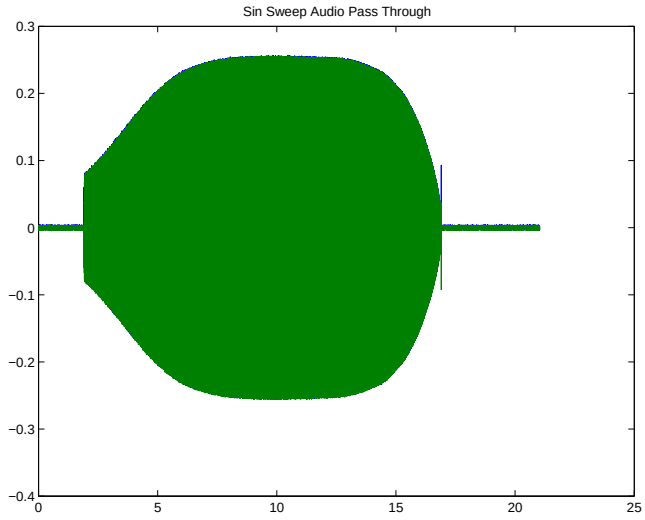

Figure 4: Time Domain Plot of Audio Pass Through

The audio crossover was applied with a crossover frequency of 1000Hz and the output of each of the two channels plotted separately as per Figures 5 and 6. It can clearly be seen that channel one presents a high pass filter and channel two presents a low pass filter, when comparing Figures 3, 5 and 6.

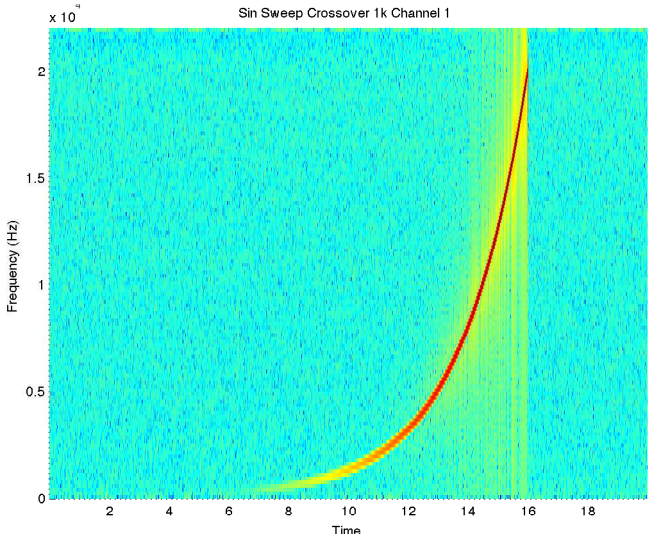

Figure 5: Spectrogram of Channel 1 Output with Crossover of 1000Hz

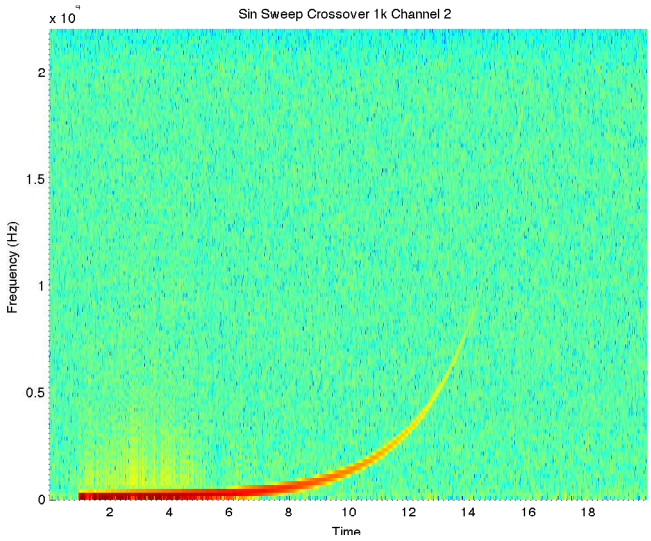

Figure 6: Spectrogram of Channel 2 Output with Crossover of 1000Hz

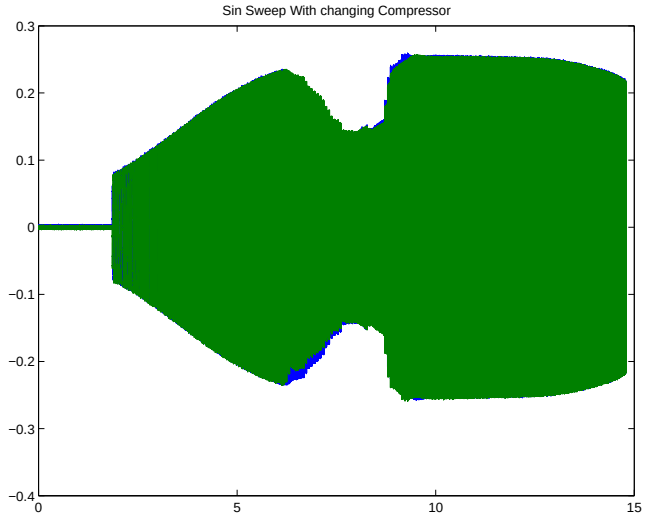

Figure 7: Time Domain Plot of Compressor with Changing Threshold through audio

6 Conclusion

In conclusion, as audio system controller had been produced, that will operate on a very basic one in two out manor. It provides a series of hardware, real-time functionality that would be expected from such a piece of hardware and some of functionality has been visualised to prove its effectiveness.
The following attributes have been implemented as part of this paper

Linkwitz Riley Second Order Crossover with hardware control of crossover frequency and independent gain control for each output channel.

Parametric Equalisation Two cascaded filters per output channel, with hardware control of frequency, quality factor and gain.

Compressor/Limiter over audio input, with hardware control of threshold, ratio, make-up gain, attack and release.

Delay with hardware control of delay per output, for 0s to 0.5s.

Through testing of the BBB, there were limited audio glitches and under-runs presented, except with the delay. A fractional delay was implemented with a reduced delay speed, however, this delay will still produce audible audio glitches. There are also some occasional issues with the compressor producing distorted noise for a short period of time.
There is clearly significant further work that could have been implemented as part of this project, however this further work is limitless up to the point where a professional audio crossover has been produced, and such pieces of hardware and software can cost multiple thousands of pounds. As such it is felt that the work undertaken as part of the paper was was of a large scope and further work would have been beyond the scope of this paper.

6.1 Further Work

As suggested in Section 6, there is significant further work that could have been undertaken as part of this project, the a summary of some of this further work has been presented in this section.

- Smoothing all the analog inputs with a low pass filter.
- Further work on the fractional delay, using polynomial interpolation, rather than linear interpolation.
- Initialise all control values to reasonable start values.
- Implement compressor knee
- Implement RMS detector rather than Peak detector on compressor
- Implement Different Crossover Filter Types eg. Bessel, Butterworth, Linkwitz Riley
- Implement Different Crossover Filter Orders
- Allow user controllable delay in ms or m. Currently in samples.
- Implement audio thread in fixed point, as input is in fixed point
- Speed Up audio thread with VPU and assembly implementation.
- Allow for control values to be read from a file on startup.
- Allowing system controller settings to be saved for recall later.
- Use of rotary encoders for hardware control, to allow infinite control and simplify the hardware thread
- Provision of a basic display, to allow user feedback on a hardware level.
- Inclusion of Noise Gates, as per original specification.

References

[1] John Borwick. *Loudspeaker and headphone handbook*. Taylor & Francis, 2001.

[2] Martin Colloms. *High performance loudspeakers*. John Wiley & Sons, 2013.

[3] Andrew Mcpherson. ECS732 Real Time DSP. University Lecture Material, 2014.

[4] Sanjit Kumar Mitra and Yonghong Kuo. *Digital signal processing: a computer-based approach*, volume 2. McGraw-Hill New York, 2006.

[5] Dave Moffat. Assignment 3 - final project proposal. *ECS732, Real Time DSP Coursework*.

[6] Bernard Mulgrew, Peter M Grant, and John Thompson. *Digital signal processing: concepts and applications*. Macmillan Press, 1999.

[7] Will Pirkle. *Designing Audio Effect Plug-Ins in C++: With Digital Audio Signal Processing Theory*. Taylor & Francis, 2012.

[8] Josh Reiss. ECS730 Digital Audio Effects. University Lecture Material, 2014.

[9] Josh Reiss and Andrew McPherson. C4DM standard audio effect plugins. VST Plugin, 2014.

[10] tutorialspoint.com. qsort() - c function library example, 2014. URL `http://www.tutorialspoint.com/c_standard_library/c_function_qsort.htm`.