# Building a digital crossover for the BeagleBone Black and Bela

Iuliu Teodor Radu

## Introduction

Designing a digital crossover network on the BeagleBone Black requires a few steps. Firstly, it is important to note that there are two ways of designing a digital filter. One is to define poles and zeros — places of maximum and minimum amplification respectively — at a particular frequency on the z-plane. The other is to take an analogue design of a filter and translate it into digital design. I will be using the latter as I am required to implement a high pass and a low pass 2nd order Butterworth filter. In order to do this, the bilinear transformation is needed in order to map the Butterworth filter's transfer function from the (analogue) s-plane into the (digital) z-plane. At this point, it is important to acknowledge that the bilinear transformation yields a non-linear relationship between analogue frequencies $\omega_a$ and digital frequencies $\omega_d$ (Roberts, 2003-2006). The specification of the program applying the 2nd order Butterworth filters to an input requires that the cutoff frequency be user selectable. As a result, it is important to pre-warp this frequency before using it to compute the coefficients for the Butteroworth filters' difference equations. The following substitution needs to be made to pre-warp the analogue angular frequency $\omega_a$ :

$$\omega_a = \frac{2}{T}\tan\frac{\omega_d T}{2}$$

where T represents the digital sampling period. For clarity's sake, I would like to mention that T, the sampling period is the inverse of the sampling frequency:

$$\text{i.e. } T = \frac{1}{f_s}$$

Within the program, it is also important to convert the user provided frequency from Hz to radians i.e. $f_{rad} = f_{hz} \times 2\pi$ before performing the pre-warping. The program will only be dealing with the difference equation and not the bilinear transformation.

## Finding the 2nd order low pass Butterworth filter difference equation

The 2nd order low pass Butterworth filter transfer function is as follows:

$$H(s) = \frac{\omega_a^2}{s^2 + s\sqrt{2}\omega_a + \omega_a^2}$$

where the Q factor has been simplified into $\frac{1}{\sqrt{2}}$ (Zumbahlen, 2008). Performing the bilinear transformation, H(s) = H(z), we have to make the following substitutions

$$s = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} \text{ and } \omega_a = \frac{2}{T}\tan\frac{\omega_d T}{2} \text{ for the pre-warping}$$

Solving this looks as follows:

$$H(z)=\frac{\left(\frac{2}{T}\right)^2\left(\tan\frac{\omega_d T}{2}\right)^2}{\left(\frac{2}{T}\right)^2\left(\frac{1-z^{-1}}{1+z^{-1}}\right)^2+\sqrt{2}\left(\frac{2}{T}\right)^2\left(\frac{1-z^{-1}}{1+z^{-1}}\right)\tan\frac{\omega_d T}{2}+\left(\frac{2}{T}\right)^2\left(\tan\frac{\omega_d T}{2}\right)^2}$$

**NB:** $\left(\frac{2}{T}\right)^2$ **cancels out**

Let $\tan\frac{\omega_d T}{2}=\tau$ since we do not have the cutoff frequency until the user provides it. I have picked Tau arbitrarily for this substitution and it has absolutely no relation to time.

$$\frac{\tau^2}{\frac{\left(1-z^{-1}\right)^2}{\left(1+z^{-1}\right)^2}+\frac{\sqrt{2}\left(1-z^{-2}\right)\tau}{\left(1+z^{-1}\right)^2}+\frac{\tau^2\left(1+z^{-1}\right)^2}{\left(1+z^{-1}\right)^2}}=\frac{\tau^2+2\tau^2z^{-1}+\tau^2z^{-2}}{1+\sqrt{2}\tau+\tau^2+z^{-1}\left(2\tau^2-2\right)+z^{-2}\left(1-\sqrt{2}\tau+\tau^2\right)}=\frac{Y(z)}{X(z)}$$

Now we can use the time shifting property of the z-transform (i.e. that $z^{-k}X(z)$ in the Z-domain corresponds to $x[n-k]$ in the time domain) in order to determine our difference equation (Kundur 2013).

$$\therefore y[n]=x[n]\frac{\tau^2}{1+\sqrt{2}\tau+\tau^2}+x[n-1]\frac{2\tau^2}{1+\sqrt{2}\tau+\tau^2}+x[n-2]\frac{\tau^2}{1+\sqrt{2}\tau+\tau^2}-y[n-1]\frac{2\tau^2-2}{1+\sqrt{2}\tau+\tau^2}-y[n-2]\frac{1+\tau^2-\sqrt{2}\tau}{1+\sqrt{2}\tau+\tau^2}$$

**Finding the 2nd order high pass Butterworth filter difference equation**

The 2nd order high pass Butterworth filter transfer function is as follows:

$$H(s)=\frac{s^2}{s^2+s\sqrt{2}\omega_a+\omega_a^2}$$

where $\frac{1}{\sqrt{2}}$ represents the simplified Q factor of the filter (Zumbahlen, 2008). By repeating the method above, we determine the following difference equation:

$$y[n]=x[n]\frac{1}{1+\sqrt{2}\tau+\tau^2}+x[n-1]\frac{-2}{1+\sqrt{2}\tau+\tau^2}+x[n-2]\frac{1}{1+\sqrt{2}\tau+\tau^2}-y[n-1]\frac{2\tau^2-2}{1+\sqrt{2}\tau+\tau^2}-y[n-2]\frac{\tau^2-\sqrt{2}\tau+1}{1+\sqrt{2}\tau+\tau^2}$$

**Design and implementation**

Most of the equations above will not appear in the software implementation. After everything has been solved as above, it becomes apparent that there are lots of recurrent parts of these equations that could be extracted into variables. This is the very reason why I created the Tau variable above. This is reflected in the code as I have created a variable called ‘tangent’. Similarly, upon inspection, it becomes apparent that the denominators of all the coefficients are the same. I have similarly saved this into a variable entitled ‘denominator’ within the code. Also, it is apparent that in both the low pass filter and high pass filter the b0 and b2 coefficients are equal. Therefore, in the code it will be sufficient to compute just one of them.

In order to save the current, last, and second last inputs and outputs, I have used circular buffers so as not to move things in memory too much. There are three of these buffers, one for the low pass filter results, one for the high pass filter results, and one for the input, each of which consists of three floats. A common buffer pointer keeps track of all three buffers. As this wraps around the buffer length size (by use of the modulo operator), I can reach the previous sample on any of the three buffers by adding two, or I can reach the second last sample by adding one to this buffer pointer.

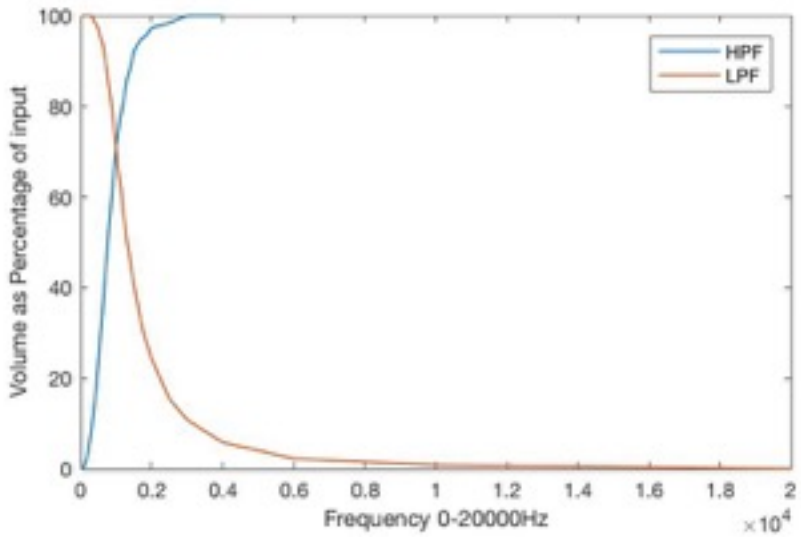
Upon entering the audioFrames loop inside of render, I started by emptying the ‘current’ buffer location. Then, I merged the stereo channels into a mono mix by adding them together and placing them inside the ‘current’ input buffer location. I then computed the difference equations for each of the filters and placed them in their respective buffers.

I decided to use the inbuilt oscilloscope to test my application in conjunction with an online sine wave generator by Tomasz P. Szynalski.

**Testing**

Initially, I checked all of the coefficients against the butter() function in Matlab. In the setup() function, I am printing all of the coefficients in order to check this. Initially I made mistakes with the algebra in the bilinear transform, but this crosschecking helped me to zero in on my mistakes and eventually get it right. Having tested the low pass and high pass Butterworth filters at different frequencies below and above the 1000hz cutoff frequency, I have assembled the following graph.

**Butterworth Filter Frequency Response at a 1000Hz cutoff frequency**



The testing has been done by sampling sine waves from within the program itself rather than by plugging in a sine synthesiser into the Bela. The reasons for this is that it appears that the Bela's input system attenuates frequencies under about 180Hz and as a result doesn't give an accurate representation of the filter.

**The Linkwitz-Riley plot twist**

In order to make a fourth order Linkwitz-Riley filter, it is necessary to cascade two second order Butterworth filters (Linkwitz, 2009). This means to square the transfer function. I have computed the Linkwitz-Riley filter coefficients in terms of the Butterworth filter coefficients as follows:

Let the Butterworth filter coefficients be equal to $b_0, b_1, b_2, a_0, a_1, a_2$, where $a_0 = 1$ (the current output coefficient). Square the Butterworth transfer function:

$$H(z)^2 = \frac{\left(a_0 + a_1 z^{-1} + a_2 z^{-2}\right)^2}{\left(b_0 + b_1 z^{-1} + b_2 z^{-2}\right)^2} = \frac{a_0^2 + z^{-1}\left(2a_0 a_1\right) + z^{-2}\left(2a_0 a_2 + a_1^2\right) + z^{-3}\left(2a_1 a_2\right) + z^{-4}\left(a_2^2\right)}{b_0^2 + z^{-1}\left(2b_0 b_1\right) + z^{-2}\left(2b_0 b_2 + b_1^2\right) + z^{-3}\left(2b_1 b_2\right) + z^{-4}\left(b_2^2\right)}$$

The five coefficients of the inputs and outputs of the Linkwitz-Riley filter become evident:

$$a_0' = 1\,;\ a_1' = 2a_1\,;\ a_2' = 2a_2 + a_1^2\,;\ a_3' = 2a_1 a_2\,;\ a_4' = a_2^2\,;\ b_0' = b_0^2\,;\ b_1' = 2b_0 b_1\,;\ b_2' = 2b_0 b_2 + b_1^2\,;$$
$$b_3' = 2b_1 b_2\,;\ b_4' = b_2^2$$

Within the code, the variable names have been aptly named eg. 'gLPFIrB0' etc to distinguish the Butteroworth filters from the Linkwitz-Riley, and between the low pass and high pass filters. Upon having a look at the oscilloscope, we can tell that it is working because now both sine waves seem to be in phase, even though they are actually 360 degrees out of phase.

**Usage**

To select between the Butterworth and Linkwitz-Riley filters it is necessary to pass in a -l option on the command line. This indicates that it is a Linkwitz-Riley filter that should be used. If this option is not set, this indicates to the program to use a Butterworth filter. The -f option followed by an argument representing the cutoff frequency in Hz should be used if you do not want the program to default to a cutoff frequency of 1000Hz.

**References**

Roberts, Stephen (2003-2006), *Signal Processing and Filter Design,* Lecture Notes, Oxford Department of Engineering Science

Kundur, Deepa (2013), *The Z-Transform and its Properties,* Lecture Notes, University of Toronto

Szynalski, Tomasz P., *Online Tone Generator,* http://www.szynalski.com/tone-generator/, last accessed 11/02/18.

Zumbahlen, Hank (2008), *Linear Circuit Design Handbook*, Ch. 8. Newnes/Elsevier.