DESC9115 Lab Report 2:

Multiband Flanger

By Tony Basa: 430574977

Tony Basa
430574977

Abstract:

Presented in this report is a digital multiband effect processor. The prototype analyzed in this report is written as a matlab script. The script splits the input signal into 3 bands highs, mids, and lows and allows each band to be processed individually to achieve a unique sound that is intended for use in music production and mixdown. The effects that are currently apart of the script are flanging effect. The matlab script has an easy to use interface for inexperienced users of matlab to process their input signals.

Introduction:

In audio production it can be very useful to process audio signals in different frequency bands instead of a broadband signal. An example of this scenario is when you want to add a flanger effect to drum mix, however you only want to apply this flanger to the cymbals. How will you do this? By splitting the signal into individual frequency band and then affecting each band separately with the desired effect. This report looks at a proposed multiband flanger that aims to solve the previously stated problem. The effect is in the form of a matlab code and this report will walk the reader though it explaining how it works.

Processor design:

The processor consists of two main components groups: a three band filter section that splits the input signal into a high frequency, mid frequency and low frequency components at preset crossovers frequencies, and a modulation effect section that processes the separated bands with a flanger effect.
Filter section:
Before the signal can be processed by the desired effect it must first be split into the low, mid and high frequency bands. Using the *fdesign* function in matlab three filters were created a lowpass filter, and bandpass filter and a highpass filter.

Low pass filter
The lowpass filter was designed using the matlab function *fdesign.lowpass.* This function creates a lowpass filter with user defined filter parameters that define the filter. The default string entries are pass band frequency, stop band frequency, pass band ripple & stop band attenuation. The filter designed in this script is as follows:

```
  PASSHZ_LOW = 250; %sets the passband frequency for the
LPF in Hz
  STOPHZ_LOW = 300; %sets the stopband frequency for the
LPF in Hz
  RIP_LOW = 1;      %sets the passband ripple for the LPF
  STOPDB_LOW = 50;  %sets the stopband attenuation for
the LPF in dB
```

The stop band and pass band were chosen based on popular multiband effect crossover frequencies (White, Robjohns 2002). White (2002) suggests a low frequency band of around 175Hz for stereo buss inserts. In this processor the frequency has been raised for 2 reasons; the first being that subtle flanging (slow rates and small width) at low frequencies are not perceived very well on consumers playback systems. The second reason was to create a smaller Q at the crossover band (fig.1). A .1dB of ripple in the pass band was set so the filter will not impart any colour to the input signal. 60db of stop band attenuation was set so the ripples after the stop band would be inaudible.

High pass filter:

The high pass filter was designed using the *fdesign.highpass* function built into matlab. Using the same string set as the *.lowpass* function a high pass filter was designed with a stop band frequency of 3500Hz and a pass band of 4000Hz. Again the pass band frequency was based of the popular multiband crossover frequency (White & Ronjohns 2002). It is recommended that 6Khz be the crossover frequency between mid and high frequency bands if the users want to achieve a sense of air in the mix. For the purpose of this processor the high frequency band has been lowered so the differences in the flanging effect can be more distinct. The function is as follows:

```
     HIdesign = fdesign.highpass('Fst,Fp,Ast,Ap',
STOPHZ_HI, PASSHZ_HI, STOPDB_HI,...
      RIP_HI, fs);
   %Creates highpass filter using default highpass string
elements
  HPF = design(HIdesign,'IIR');
   %Implements filter design to HPF
```

The filter is implemented with an IIR design using the *design* function. This design method was chosen over the FIR method because it achieved more desirable slope with less ripple. (fig2.)

Band Pass filter:

The band pass filter was designed using the *fdesign.bandpass* function. This function is similar to the previously used *fdesign.* as it gives the user the ability to input filter parameters to design a specific filter. In this case the filter type is band pass and has a default string input of frequency stop low, frequency pass low, frequency pass high, frequency stop high, stop band attenuation for the low frequency, stop band attenuation for the high frequency and a pass band ripple limit. The frequency limits were set based on the low and high pass filters. (fig.3). For consistency all filters used an IIR method. This method provided steep pass band attenuation and very little pass band ripple.

Modulation section:

The next step of the script implements a flanger function on each frequency band of the input. The flanger function used in this script is based of a guitar flanger effect pedal. Written by Luster (2013), the function uses a sawtooth wave shaped low frequency oscillator (LFO)instead of a sine wave

shape. This produces an audible effect closer to that of a classic guitar flanger sound.
The function processes the input at four times the sampling rate for a higher quality audible output. Below is a section of code from the flanger function.

```
minDelaySamp=ceil(delay*fsn/1000); %convert to msec, then
samples
maxDelaySamp=ceil((delay+width)*fsn/1000); %convert to
msec, then samples
n=(1:length(in)+maxDelaySamp)'; %how long to extend in by
for LFO
LFO=sawtooth(2*pi*rate/(fsn)*n,.5); %sawtooth more
commonly used in flangers
delayTimeSamples=(delay+width/2+width/2*LFO)*fsn/1000;
```

The first line derives the minimum delay length, based of the user set *delay* variable, is samples. The second line derives the maximum delay length using the user-defined variable of *width.* The function then defines the total length needed for the LFO. The sawtooth function in the fourth line creates the LFO using the user defined *rate* variable. The .5 values at the end of the equation specifies the offset of symmetry of the triangle wave (matlab).

The final steps of script gives the user the option to change the individual frequency band levels before they are recombined back into mono. This feature adds flexibility to the processor as it can be used to shape the over all mix.

User interface:

The matlab user interface can be daunting and at times confusing if the user is unfamiliar with the software. For this reason the script contains user-friendly interactions. The user is prompted to enter parameter values for the flanger effect in each of the frequency bands. The display also gives limits to what values may be entered:

```
  display= '\n mid Band Flange Width (ms) 0.1-10\n\n >';
  mwidth= input(display);

  display= '\n mid Band Flange Rate (HZ) 0.05-5\n\n >';
  mrate= input(display);

  display= '\n mid Band Flange Mix 0-1\n\n >';
  mmix= input (display);
```

This makes it easy for first time users to get workable results with their input signals. However the drawback is that this interfaces makes it cumbersome to change parameters quickly and experienced matlab users might find this feature frustrating.

Conclusion:

If this product is picked up for development many more features can be implemented. An example of this could be to implement a compressor over each frequency band, or a phaser. More filters could be added and the signal could be split into more frequency bands allowing for more effect processing to be done to the signal.

Fig.1

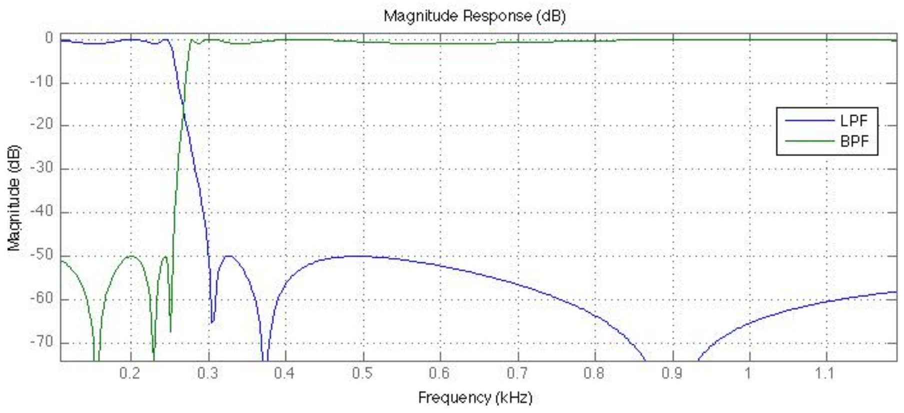


Fig.2

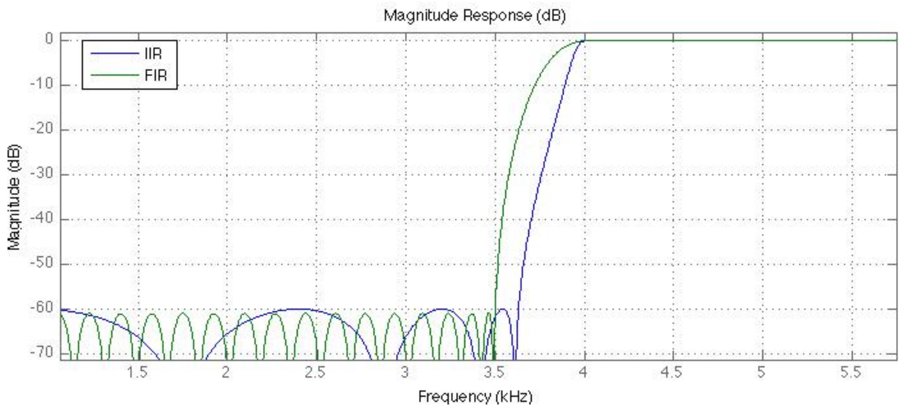
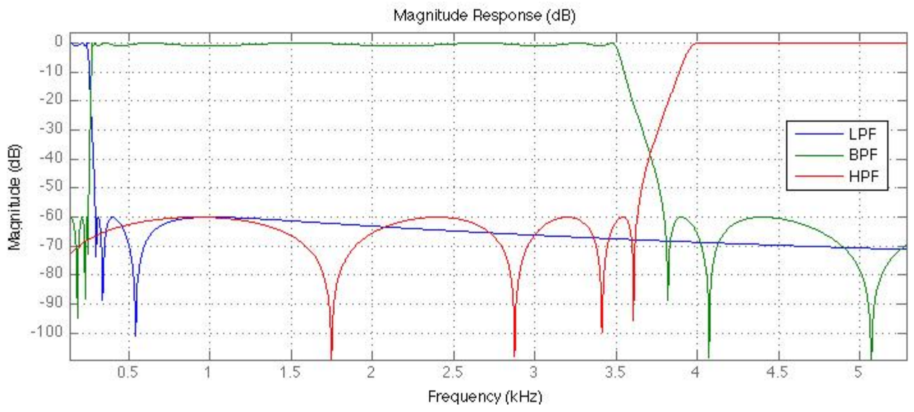


Fig.3



Reference List:

White, P Ronjohns, H 2002, Multi-Band workshop, *Sound on Sound mag, August 2002.*

Luster 2013, Flanger Matlab function, https://github.com/luster/guitar-effects/blob/master/matlab/flanger.m