Problem Set 6: Real-time Filtering of WAV Files

Please send back to me via Brightspace

- A zip archive named as
  `PS06_First_Last.zip`
  Where `Last` is your last name, `First` is your first name, and the archive contains
    - the C code files that implements all aspects of all problems.
    - All requested plots (spectrogram and spectrum)

Total points: 100
See point allocation below.

References
libsndfile reference
http://www.mega-nerd.com/libsndfile/

libportaudio reference
http://www.portaudio.com/

The following URLs provides a good reference for C language library function usage:
https://cplusplus.com/reference/clibrary/

Files in zip archive
You are given the following complete C code files:

- filter.h
- fir_filt_coef.h
- pautils.c
- pautils.h

You are given partial C code files that you must complete:

- filter_file.c
- filter_block.c

You are given signal files that you will use to test your code:

- signals/chirp.wav
- signals/noise.wav
- signals/trilogy.wav

You are given Bash script **build.sh** that compiles and links the programs

Your program will use these libraries:
sndfile library to read WAV files
PortAudio library to play audio data to speaker using a callback function

Program Overview

The C-code files filter_file.c and filter_block.c filter a WAV file using an FIR filter. These are two distinct programs you must complete. The data structures for the filter are in filter.h and the filter coefficients are in fir_filt_coef.h, and these files are common to both programs.

You must add to the code in filter_file.c and filter_block.c by filling in code under the comment `//Your Code Here`.

First problem: Off-line filtering

The program filter_file.c

- Parses command line
- Opens input and output wav files, allocates storage, reads the input signal
- Filters the signal from input buffer to output buffer
- Writes the output wav file and closes all files

Second problem: Real-time filtering

The program filter_block.c

- Parses command line
- Opens input and output wav files, allocates storage, reads the input signal
- Starts PortAudo
- The PortAudio callback filters a series blocks of the input buffer to blocks of the output buffer and also plays the blocks out via the PortAudio output block buffer
- When real-time processing is cone, writes the output wav file and closes all files

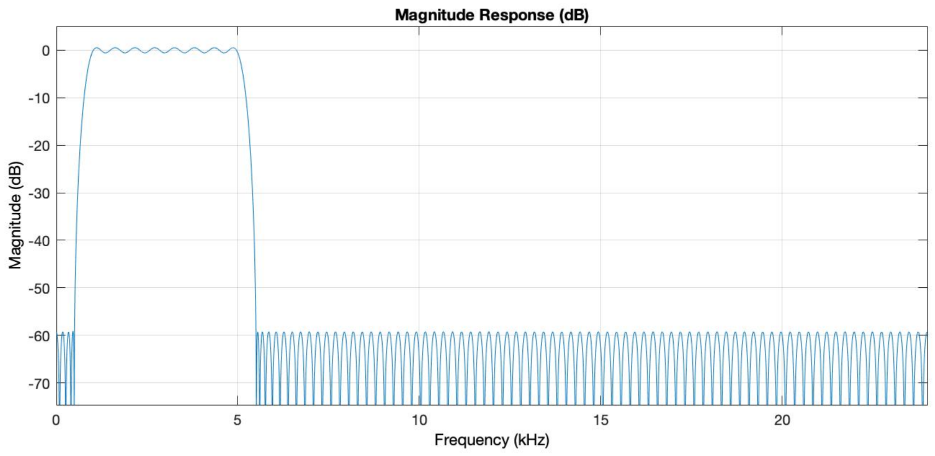
Demonstrate results

- Submit plots showing filtered output signals in time-domain, in time-frequency domain (as spectrograms) and in spectral magnitude domain.

Both FIR filtering programs filter only single-channel signals. The code structure can be generalized to multi-channel, but that is not a requirement of this assignment.

The Filter

The filter is a 191-tap (coefficient) finite-impulse-response (FIR) filter. The filter coefficients are in fir_filt_coef.h. The magnitude response of the filter is:



It is a band-pass filter with passband from 500 Hz to 5000 Hz and a stop-band attenuation of 60 dB.

(50 points) Problem 1 – filter_file.c

Parse command line

Your program should have the following command line usage:

```
./a.out in_file.wav out_file.wav
```

Where

`in_file.wav` is the input signal file
`out_file.wav` is the output signal file

Parse the command line. If parsing fails, print an error diagnostic and exit.

Open input and output files

Input

- Open the input WAV file using sf_open() using the SFM_READ option. This will read the WAV header into the `sndfile` structure.
- Print information about the WAV file (number of frames, number of channels, sampling rate).
- Write selected information to the PortAudio callback `buf` structure.

Output

- Open the output WAV file using sf_open() using the SFM_WRITE option.
- Set output sfinfo structure values to be the same as input file (samplerate, channels and format)

Allocate storage for input file and read file

- Allocate storage sufficient to store the entire input WAV file audio signal.
- Read the WAV file audio signal into the buffer.
- Close the WAV file.

Allocate storage for output file

- Allocate storage sufficient to store the entire output WAV file audio signal (input file frames + filter length – 1).

Filter signal

- Filter signal in the input buffer to produce the output buffer. This can be done in two nested for() loops, as shown here. See lecture slides for more information on the specific code for filtering.
```
for (int n=0; n<N+M-1; n++) {
     for (int k=0; k<M; k++) {
          … //filtering code
     }
}
```

After all processing, write output file, free storage and close file

- Write all values in output buffer to output WAV file.
- Free allocated storage
- Close output WAV files

(30 points) Problem 2 – filter_block.c

This problem has considerable code structure in common with filter_file.c. Once you get filter_file.c working, just copy these code blocks to filter_block.c:

- Parse command line
- Open input and output files
- Allocate storage for input file and read file
- Allocate storage for output file
- After all processing, write output file, free storage and close file

What differs in this program is that all filtering is done in real-time in the PortAudio callback. This problem has the following new elements (which are already present in the file):

- Initialize the PortAudio data structure
- Start PortAudio
- After all processing, stop PortAudio

Filter audio data in PortAudio callback

- Filter the single-channel buffer icbuf[] using both the current block and a portion of the previous block (which is in state[] in the callback data structure).
- After filtering, update the state[] buffer by copying the last `num_b-1` (`or M-1`) samples from icbuf[] to state[].
- Copy the processed section of ofbuf[] to the portaudio output buffer

See lecture slides for more information on the specific code for filtering.

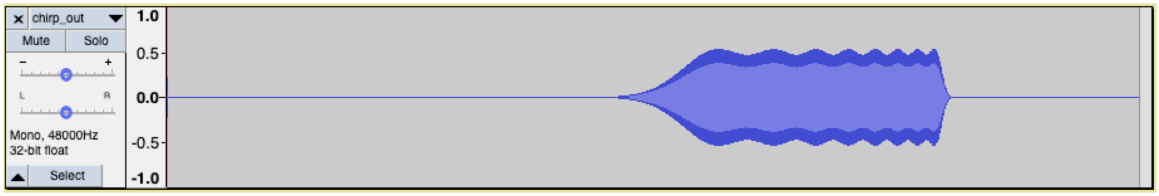
(20 points) Test your program and submit plots

Execute filter_file (filter_block would also work) as

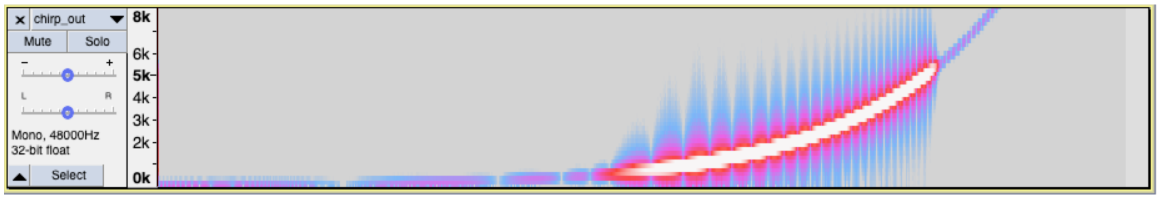
```
./filter_file signals/chirp.wav chirp_out.wav
./filter_file signals/noise.wav noise_out.wav
```

The chirp.wav signal is a logarithmic tone sweep, from 100 Hz to 10 kHz.

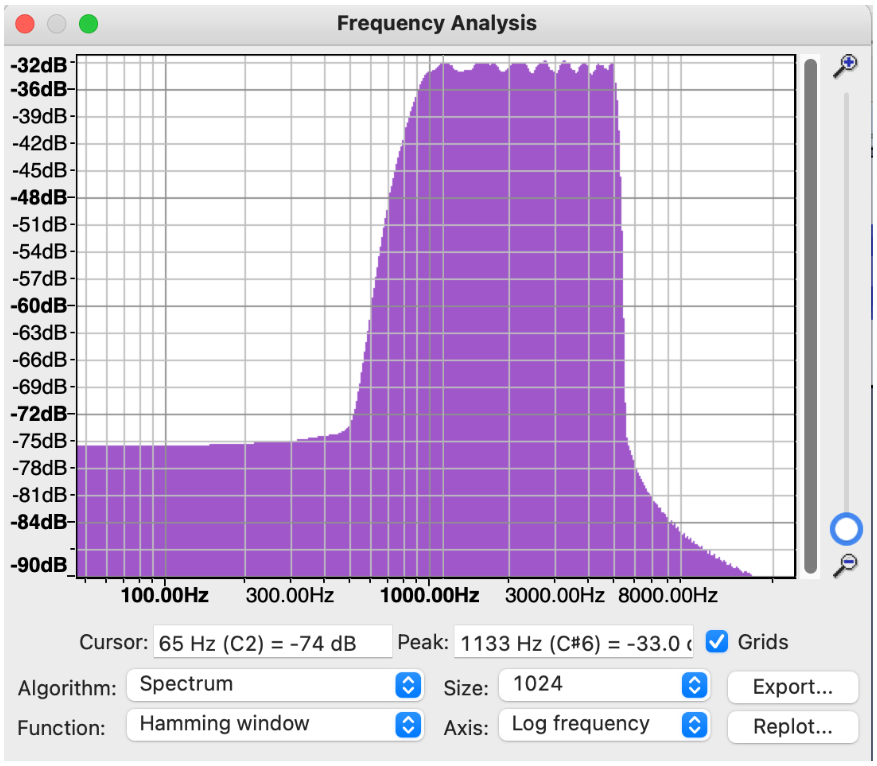
- Submit a plot of the waveform of chirp_out.wav, which should be something like the plot below. Since the bandpass filter is 500 Hz to 5 kHz, the output is zero (-60 dB) in the intervals before the chirp reaches 500 Hz, and after the chirp goes beyond 5 kHz.



- Submit a plot of the spectrogram of the chirp_out.wav signal, which should look something like this. Bright (white) region is high signal energy, indicating the filter bandpass region.



- Submit a plot of the magnitude spectrum of the noise_out.wav signal, which should look something like this. Since noise.wav is broadband white noise, the filtered noise signal clearly shows the bandpass region of the filter.



Cursor: 65 Hz (C2) = -74 dB Peak: 1133 Hz (C#6) = -33.0 d ✓ Grids

Algorithm: Spectrum Size: 1024

Function: Hamming window Axis: Log frequency