Extra Instructions and Hints for EE3221 Week 8 Lab – Dr. Durant – 2018-01-25, 18:51

- The assignment is posted at https://faculty-web.msoe.edu/prust/EE3221/RTDSP_Lab5.pdf
- You can choose the level of completion on this lab. The 3 options build on each other. You'll want to accomplish them in order, although you only need to demonstrate and submit your final version.
    - Earn up to 80% by successfully removing the interfering tone from your file with one interfering tone using the DSP hardware.
    - Earn up to 100% by doing the same for your file with 3 interfering tones.
    - Earn up to 120% by researching "Direct Form II" and "biquads" as ways to efficiently implement IIR filters and using 3 Direct Form II biquads to solve the 3-tone problem on the DSP hardware.
- The interfering tones are very loud. Take care with your volume settings, especially with earphones.
- Earbuds or similar are recommended. The speakers on your laptop likely reproduce frequencies down to 300 Hz acceptably, are noticeably quiet at 250 Hz, and have virtually no audible output at 200 Hz. Many headphones and earbuds can acceptably reproduce low frequencies down to 30 Hz, so headphones are preferred for this lab.
- You can find your audio files at https://faculty-web.msoe.edu/durant/courses/ee3221/l08/
    - NN_1_username.wav contains music with one loud, interfering tone
    - NN_3_username.wav has 2 additional loud, interfering tones added for a total of 3
- The interfering tones are all in the audible range of 200 Hz to 3 kHz.
- Confirm that you can remove the tone(s) in MATLAB before proceeding to the real-time hardware.
- Read audio with `[x, fs] = audioread('filename.wav');`
    - Type `doc audioread` and similarly for any function that you want or need to learn more about.
- Save audio with `audiowrite('filename_clean.wav', y, fs)`
- Use MATLAB's FFT function to efficiently calculate the DFT of your signal.
    - To plot the magnitude spectrum, take the absolute value of these complex results.
    - Then convert to decibels.
    - Recall that the DFT frequency values are $\omega_k = 2\pi k/N$. Convert the frequency scale to hertz for graphing using the definition of digital frequency.
        - `k = 0:(N-1);`
        - $2\pi/N$ is the fundamental frequency. In hertz, this is 1/T, where T is the duration of the signal, which is 11.96 s. So, the fundamental frequency, which is also the resolution, is 0.0836 Hz.
- To remove 3 tones, you can
    - Design and separately apply 3 filters using `filter()`.
    - Or, you can combine all 3 filters into a 6th-order filter by convolving the coefficients, which multiplies their z-domain representations: (e.g., `a = conv(conv(a_w0,a_w1),a_w2);`)

- Matlab normally only shows you a few decimal places of precision. Tell Matlab `format long` to see greater precision, which is necessary given the high r value (poles quite close to the unit circle) necessary to meet your specification.
- Use fs = 48000 Hz throughout. Use the line_in input on the DSP hardware.
- For your C implementation, your calculations will be done in single-precision floating point (float). This happens mostly automatically in C since arithmetic operations between ints like int16_t and floating point types like float are automatically converted to the floating point type. When you assign the final float result to your int16_t output, truncation occurs, which is okay. If you are storing

previous values, though (e.g., y[n-1]), you should use float values for better precision even though they are converted to int16_t when you output them.

  - If you need 2 previous values of y, you would normally declare y to have 3 elements. Then, y[0] is for holding the current value (y(n)) and y[2] is for holding the one that is 2 samples old (y(n-2)). You would normally write y[2] = y[1]; y[1] = y[0]; when preparing to process a new sample to implement the delays.

- Do not use global variables. They are not needed here and thus are bad style.
- You will need to allocate memory to remember information between samples (e.g., past samples of x and y values). In DSP systems, when memory is needed for the life of the program and does not vary in size, it is normal to allocate memory statically (for the life of the program). Thus, in the function which needs to remember past x values, you might write `static float x[3];`.
- Although you could hard-code your IIR filter coefficients directly into your implementation for this assignment, it will generally be easier to store them in an array, for example, `static const float b[] = {1, 2.3, 4.6};`. This results in b[0], b[1], and b[2]; note how this matches our notation from class, but differs from Matlab vector notation.

- This is ANSI C, so variables must be declared at the ***start*** of blocks of code (functions, {} within for loops, etc.). Also, this isn't C99, so if you wanted to write a for loop (may be helpful, but not required for this assignment), you can't write `for(unsigned int idx = 2;` …, but have to declare the index variable before the loop.
- You can call functions from the sample-by-sample interrupt handler I2S_HANDLER. That will allow you to separate your filter implementation(s) from your code for handling the CODEC. For example:

```
void I2S_HANDLER(void) { /* I2S Interrupt Handler */
   int16_t audio_out_chL = 0, audio_out_chR = 0, audio_chL, audio_chR;

   audio_IN = i2s_rx();
   audio_chL =  audio_IN        & 0x0000FFFF;
   audio_chR = (audio_IN >> 16) & 0x0000FFFF;

   //notch(audio_chL, &audio_out_chL); // single notch, 2-pole, 2-zero
   notch3bq(audio_chL, &audio_out_chL); // 3 notches, direct form II biquad
   audio_out_chR = audio_chR; // pass through

   audio_OUT = (audio_out_chR<<16 & 0xFFFF0000) | (audio_out_chL & 0x0000FFFF);
   i2s_tx(audio_OUT);
}
```

- Then, your notch filter would be implemented in `void notch(int16_t in, int16_t* out)` and you would store the output through its pointer: `*out = myResult;`.
- In the above program, I process the left channel and pass the right channel through the DSP hardware, which is useful for the network analyzer and basic debugging. This assumes that the input signal is present on both left and right channels.
  - You can accomplish this with a jumper block on line_in when using the Digilent network analyzer.
  - MATLAB does this automatically when you play a monaural (1-channel) sound using the sound() function. The given WAV is monaural.
- You are not required to use 2-D C arrays if you remove all 3 notches, but it might be helpful. E.g., `static const float a[][3] = { { a00,a01,a02}, {a10,a11,a12}, {a20,a21,a22} };`