Digital Filters in the Embedded World

Due: March 6th @ 11:59pm

Version 2018.01

This assignment will introduce you digital filtering in the embedded space, the chief complexity of which are designing filters that can be used with fixed point computations (instead of floating point computations) and accurately implement those filters.

Goal

The goal of this assignment is automatically detect low-frequency sound using the Photon. This will entail designing and implementing a digital filter that runs on the photon, coupled computing signal power.

- You can use either matlab or python to engineer an **IIR** bandpass filter. The filter should only keep sounds between 300 and 500 hz. Signals outside of 200 and 800 hz should have less than 5% (-26dB) pass through.
- You will then implement your **IIR** filter as a fixpoint computation on the Photon.
- The Photon will then determine if a low-frequency sound is present or not.

To facilitate grading we asked that you make a button press begin recording for 5 seconds. Please ensure your Photons are connected to `iu-devicenet`, and that your devices are sampling at 32khz.

While recording you should

1) Upload the unfiltered data as a wav file to the iot server (iot.lukefahr.org)
2) Filter the data, and upload the filtered data as a wav file to the iot server
3) Compute the signal power and use it to determine if the audio signal contains the frequencies of interest. If so, it should display this by making an LED turn on (or change color).

Grading

For grading, we will:

1) Ask you to demonstrate the frequency response of your filter in python or matlab.
2) Download your unfiltered and filtered wave files from the iot server and compare the two to evaluate the effectiveness of your filter implementation on the Photon.
3) Play sounds of 180, 440, and 820 Hz to test if your system can correctly detect signals in the desired frequency range.

The grade for this project will be based on:

- **15%** The filter design strategy, specs, and coefficients yield a filter that meet the requirements listed in the last section.
- **25%** Filter works in floating point (implemented in python/matlab)
- **50%** Filter works in fixpoint (this is graded on graduated scale that we will provide later)
- **10%** LED correctly indicates sounds within desired frequency range

Notes

Second Order Sections

It is difficult to implement fixed point IIR filters that are of more than order 2, i.e. they keep more than two history elements. If you need a filter greater than 2nd order (and you should), then you will need to break the IIR filter into "Second Order Sections". This splits the filter into multiple cascaded filters as follows:

x[n] -> IIR Filter #1 -> IIR Filter #2 -> IIR Filter #3 -> y[n]

This conversion can be done by hand, but Python and Matlab provide the `'tf2sos'` (Transfer Function To Second Order Sections) function for you. See the example code for more details.

CMSIS

Your Photon contains a ARM Cortex M3 cpu. Luckily, ARM knows that many people like to run digital filters on their cpus, so they provide a library ("CMSIS") for making that "easy". Unfortunately, the Photon does not include this library by default. However, you can download the library here:

https://github.com/ARM-software/CMSIS/tree/master/CMSIS/DSP_Lib

It contains a predefined fixed-point 32-bit 2nd order "biquad" IIR filter, called `arm_biquad_cascade_df1_q31()`. You might find it helpful.

https://github.com/ARM-software/CMSIS/blob/master/CMSIS/DSP_Lib/Source/FilteringFunctions/arm_biquad_cascade_df1_q31.c

Other Resources

- We will provide code examples in matlab and python.
- We provide a tutorial for designing a digital filter in matlab, the course lecture note are essentially the same for python.
- https://www.dsprelated.com/freebooks/filters/BiQuad_Section.html
- https://www.dsprelated.com/freebooks/filters/Direct_Form_I.html
- https://www.mathworks.com/matlabcentral/answers/230948-sos-filter-how-to-manually-apply-scale-values
-