Assignment #5

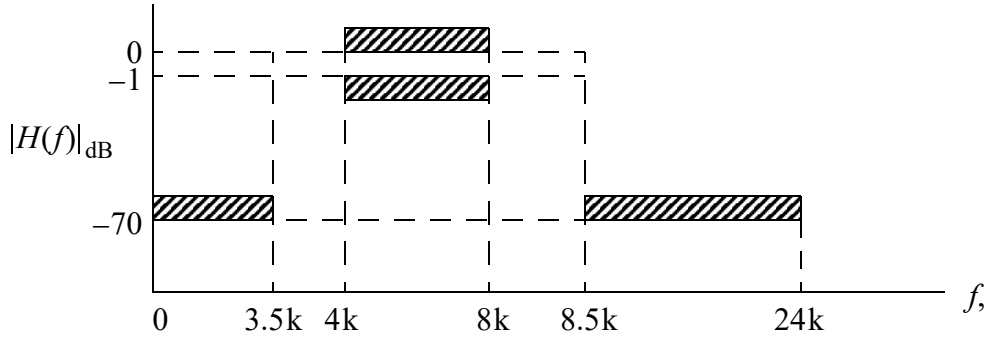
Due MondayApril 24, 2017

Make Note of the Following:

- If possible write your lab report in Jupyter notebook
- If you choose to use the spectrum/network analyzer to obtain tiff graphics, just import these graphics files into Jupyter notebook as well.

Problems: Real-Time IIR Digital Filters

1. Cascade of Biquads IIR Using Floating-Point Coefficients
   a.) Using a 48 kHz sampling rate and the main module `FM4_IIR_intr.c`, design an **elliptic** bandpass filter using Python (see examples in notes Chapter 7) that satisfies the amplitude response specifications shown below. Create a header filter file in SOS format as described in the Assignment 5 Jupyter notebook.



   b.) Provide Python design information, including magnitude and phase response plots. Your plots should use a digital frequency axis scaled to the actual sampling frequency.
   c.) Verify the filter real-time frequency response by placing the coefficients in a working cascade of biquads IIR filtering program such as described in notes Chapter 7.
   d.) For verification all you need to obtain is the frequency response magnitude in dB. The best approach is probably to use the vector network analyzer, but optionally you can drive the filter routine with the software noise generator or the noise source in the Analog Discovery. Once you have experimental data in a file, import it into your Jupyter notebook (again see the Chapter 7 sample notebook) to overlay with the theoretical response.
   e.) Time your code with `-o3` optimization using the GPIO IRQ timing pulse. Estimate the maximum sampling rate possible without loss of real-time performance? Switch to the ARM SOS routine to see how much faster it is.

2. Using the GUI parameter slider control for he Cypress FM to implement a variable center frequency notch filter of the form

$$H(e^{j\omega}) = \frac{1 - 2\cos\omega_0 z^{-1} + z^{-2}}{1 - 2r\cos\omega_0 z^{-1} + r^2 z^{-2}},\ 0 < \omega_0 < \pi$$

with $r = 0.9$. The parameter the GUI adjusts will be ω_0 or have some relation to it. Use `float32_t` for your design. Test the filter with a sampling frequency of 48 kHz, and verify using the network analyzer the ability to tune the notch around the interval $f_0 \in (0, f_s/2)$ Hz. What is the notch depth in dB when tuned to about 1 kHz? I will expect a sound demo of this in the lab. A nice test of this system is to sum an audio music source together with a single tone *jammer* from a function generator, and then see how well the tone can be suppressed from the music without coloration. With a second slider you can use r to adjust the filter bandwidth. As an audio source you can use an MP3 player/iPod, portable CD player, or Web radio.

Note: The notch filter above is really just a single biquad section. So create an SOS coefficients array as described in notes Chapter 7, such that you can change values in the array according to the needed ω_0 and r values. Then you can use the filter functions found in `IIR_filters.c/IIR_filters.h`, i.e., `IIR_sos_filt_float32()` or use the CMSIS-DSP functions `arm_biquad_cascade_df2T_f32()/arm_biquad_cascade_df2T_init_f32()`. As globals include

```
// Create (instantiate) GUI slider data structure
struct FM4_slider_struct FM4_GUI;

// IIR notch filter related variables
float32_t fs = 48000;
float32_t r = 0.9;
float32_t f0 = 12000.0;
float32_t ba_coeff1[5]; // single SOS section in place of IIR SOS header include
float32_t x, y, IIRstate1[2];

struct IIR_struct_float32 IIR1;
arm_biquad_cascade_df2T_instance_f32 IIR2;
```

In main initialize the GUI slider accordingly and the one only section biquad as

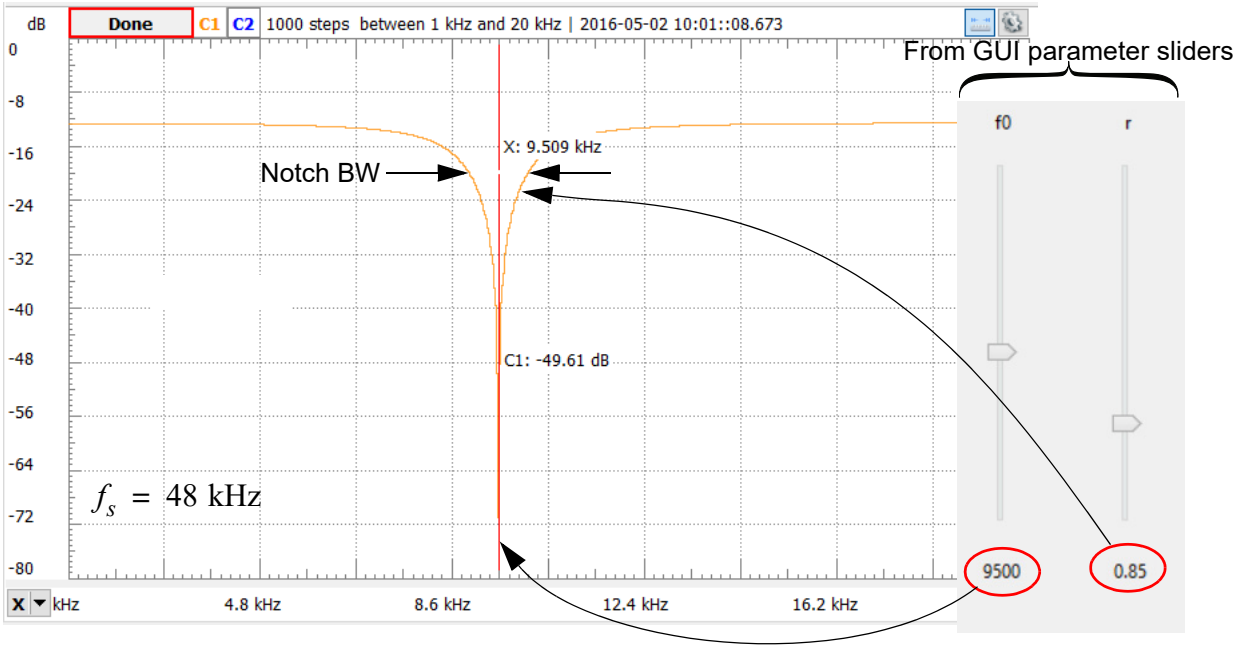
```
init_slider_interface(&FM4_GUI,460800, 1.0, 1.0, 0.0, 0.0, 12000.0, 0.9);
 ...
//Initialize IIR notch to the f0 and r values
ba_coeff1[0] = 1.0;
ba_coeff1[1] = -2.0*cosf(2*PI*f0/fs); // or use FM4_GUI.P_vals[4] for f0
ba_coeff1[2] = 1.0;
ba_coeff1[3] = -r*ba_coeff1[1]; // or use FM4_GUI.P_vals[5] for r
ba_coeff1[4] = -r*r; // or use FM4_GUI.P_vals[5] for r
```

I recommend updating the filter coefficients only when the parameter slider of interest changes. That means putting an `if()` code block inside the main `while()` loop where fol-

lowing `update_slider_parameters()`,

```
...
  while(1)
{
  // Update slider parameters
  update_slider_parameters(&FM4_GUI);
  // Reload notch parameters if P_idx is 4 or 5 (meaning f0 or r has changed)
  if((FM4_GUI.P_idx == 4) || (FM4_GUI.P_idx == 5))
  {
    ba_coeff1[1] = -2.0*cosf(2*PI*FM4_GUI.P_vals[4]/fs);
    ba_coeff1[3] = -P_vals[5]*ba_coeff1[1];
    ba_coeff1[4] = -P_vals[5]*P_vals[5];   }
  }
```

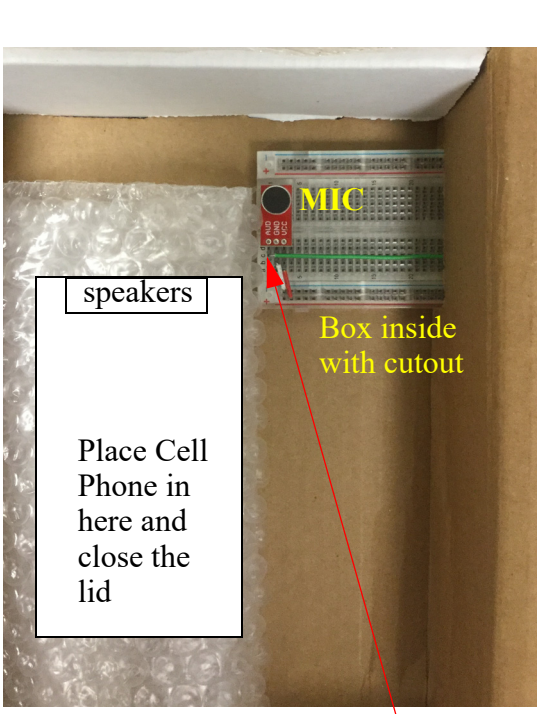
The expected results using the Analog Discovery network analyzer, take the form

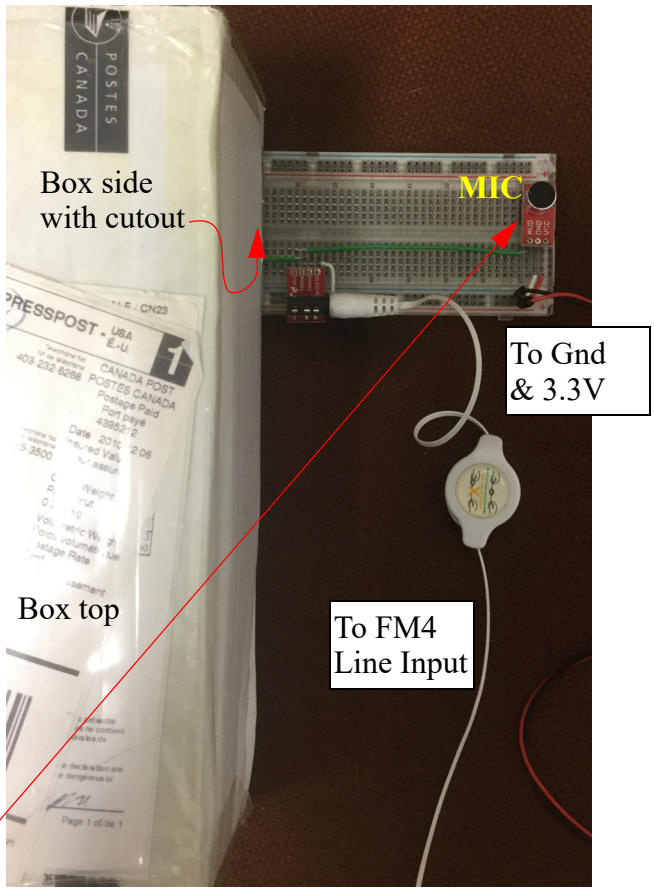


3. Adaptive filtering using CMSIS-DSP: In this problem you will extend the in-class demo found in `Assignment_Adapt_sp2017.zip`. This problem does not require writing any new code, but rather running the existing code using microphones as the input to an adaptive interference cancellation experiment.

   One microphone is located inside a box where you will place your cell phone set to play music through its built-in speakers. The second microphone is outside the box where a source of interference, a sinusoidal tone produced by a signal generator resides. Some of the interference outside the box leaks to the inside of the box.

The experimental set-up is shown below:





Signal + Interference

~Interference Alone

Using a PC speaker interfaced to a function generator or Analog Discovery, generate a sinusoidal interfering tone around 1 – 6 kHz. With music play from the cell phone monitor the error signal from the adaptive filter. With no adaptation you should see the music spectrum along with the interfering tone that leaks into the box. With the adaptive filter converged the interfering tone should be much reduced in amplitude.

a.) Measure the dB reduction in interference with and without the adaptive interference canceling turned on.

b.) Verify that as you change the frequency of the interfering tone the adaptive filter picks up the change and adapts once again to remove the interfering tone.

I will expect screen shots of waveforms and spectra explaining that the experiment is working. Giving me demo is optional, but encouraged.