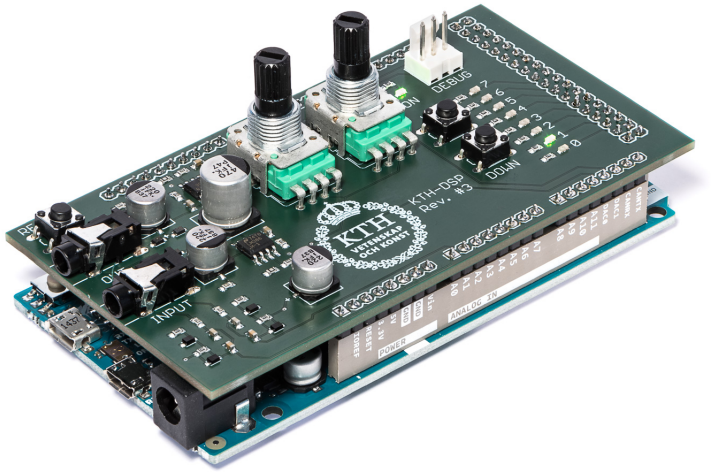
Project Assignment, HT 2018
EQ2300 Digital Signal Processing



1 Introduction

In this project, we will design low-pass and high-pass filters that will later be used to implement a two-channel audio equalizer on the Adruino Due platform, shown above. The Arduino Due has a 32-bit ARM microprocessor, the AT91SAM3X8E (SAM3X), clocked at 84 MHz. As we wish to use the equalizer for audio signals, we will choose a sampling rate of 48 kHz. This will give us $84 \times 10^6/48 \times 1^3 = 1750$ clock cycles per sample of the input signal, or rather input signals as we will wish to have stereo signals with both a left and a right channel. This is not a lot of computational power, but it will be sufficient for our project. However, as the SAM3X does not have a floating point unit, we will have to implement our equalizer using fixed point processing.

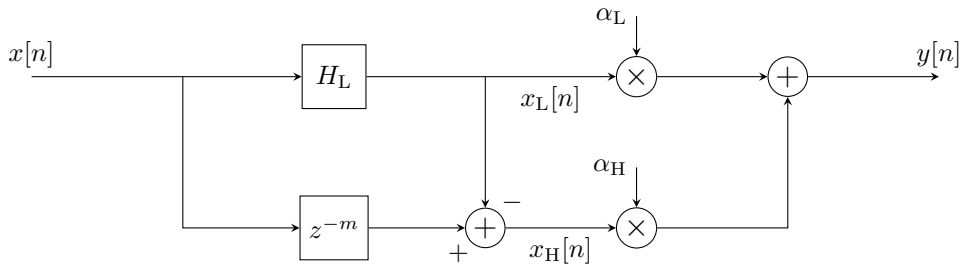
1.1 Project reporting

The project should be solved in groups of at most two students, and reported in a project report of at most 4 pages. The report should to be submitted via the course web page by **Friday November 30, 2018**. This project is only concerned with the filter design problem. The implementation of the filters are then covered by the course lab. The project is divided into 5 explicit tasks described below. It is recommended that you work with Matlab for the numerical parts of the project, and you should hold on to your Matlab code as the filters you design now will be used in the lab. If you are more comfortable working with another programming language, this will also be ok, but you may not have access to some convenient filter design functions that are part of Matlab.

Your report should be one flowing text, with an introduction to the problem, a section that contains your solution to the project, and a short conclusion. You should avoid making your project report just a list of answers to the tasks, with little of no words in between. Imagine that you are writing the report so that one of your fellow students, who have also seen this project specification, could read and reproduce your results. For this, it is important to not leave out any details that would be needed to recreate your results. In particular, **it should definitely be possible for the teacher and TAs to regenerate all plots and results in your report based on your descriptions**. An example project report will be made available on the course web to help explain what a typical report should look like. The source code for the example project report and for these project instructions (which are written in the typesetting language LaTeX) will also be made available in case you wish to take the opportunity to learn this fantasticly elegant typesetting language.

Finally, note that you may consult and discuss with people outside your project group, but your final solution and report need to be unique to your group.

2 Project specification and tasks



The basic structure of the equalizer to be designed is shown above. The component marked H_{L} is a discrete time low-pass filter that will filter out the lower frequencies of the input signal $x[n]$, and give the low-frequency signal $x_{\mathrm{L}}[n]$. The high-frequency signal $x_{\mathrm{H}}[n]$ is obtained by subtracting $x_{\mathrm{L}}[n]$ from a delayed version $x[n-m]$ of the input $x[n]$. Multiplying the two signal components with variable factors $0 \leq \alpha_{\mathrm{L}}, \alpha_{\mathrm{H}} \leq 1$ and then summing the components back together again will complete the equalizer where α_{L} and α_{H} can be used to control the amount of low-frequency and high-frequency contents in the equalized signal.

Taks 1: Verify that the output $y[n]$ will always be a delayed output of $x[n]$ if $\alpha_{\mathrm{L}} = \alpha_{\mathrm{H}} = 1$, regardless of the choice of H_{L}. Include a short proof in your report.

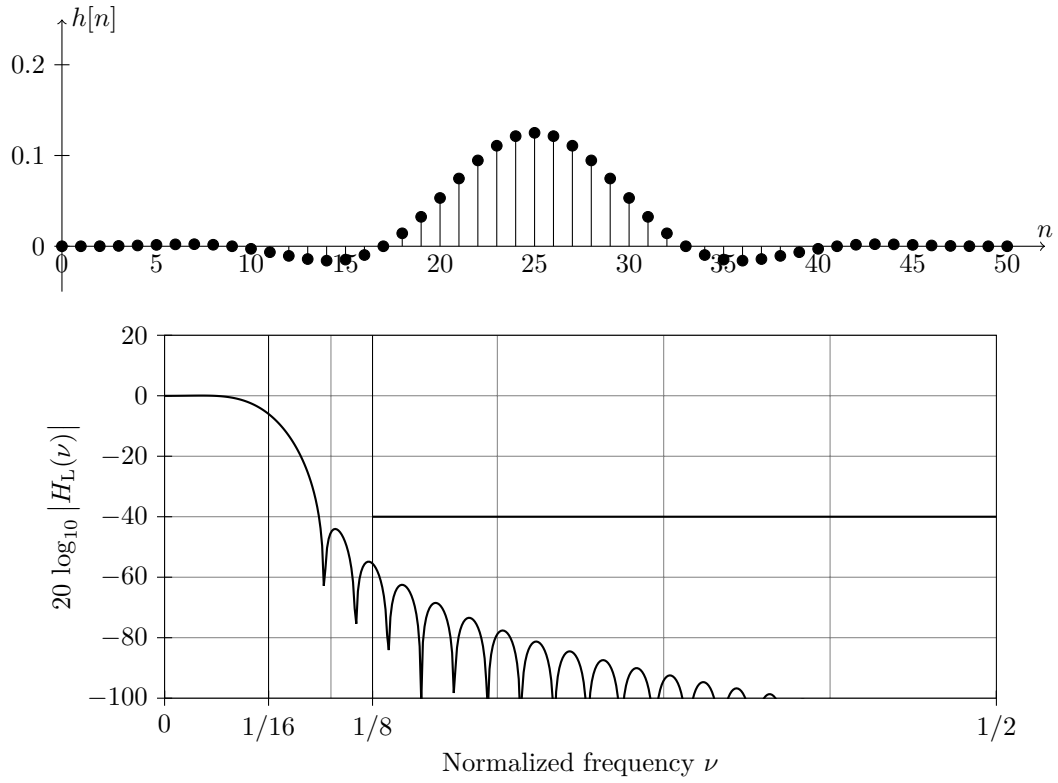
2.1 Low-pass filter design

The low-pass filter H_{L} is supposed to filter out low frequencies of the input signal $x[n]$, by attenuating the higher frequencies. Most of the power and vocals of music are contained below 3 kHz, so for this reason we will choose this as the cutoff frequency between the frequency bands. The platform can handle filters of length up to around 60 taps in the time-domain without optimization, but to be safe it is advisable to keep the filter length at or below $M = 55$ taps. You are of course encouraged to push the platform to the limit, but you will be less likely to experience problems if you play it safe.

Taks 2: Design a Type I linear phase causal FIR filter with your method of choice. The filter should have a nominal normalized cutoff frequency of $\nu_{\mathrm{c}} = 1/16 = 3/48$, corresponding to an equivalent cutoff frequency of 3 kHz for the analog signal before sampling. The filter need to have a suppression of at least 40 dB for all normalized frequencies above $\nu_{\mathrm{c}} = 1/8 = 6/48$, corresponding to an equivalent analog frequency of 6 kHz. As the filter should eventually run on the target platform, it should not have more than $M = 55$ taps, but you could design a shorter filter than that as long as you meet the specifications. You may need to design your filter with a bit of margin so that you can later quantize the filter coefficients and still meet the specification. Your final report should contain plots of both the filter's impulse response $h_{\mathrm{L}}[n]$ as well as the magnitude of the frequency response $H_{\mathrm{L}}(\nu)$ in dB scale for $\nu \in [0, \frac{1}{2}]$, see examples below.

In Task 2 it is suggested that you work numerically in Matlab with your design. The design method should preferably be one that is presented in the course, i.e., the method of windows or frequency sampling. Unless you have a reason for choosing frequency sampling, we suggest that you go for the method of windows and you may in this case be helped by the built in Matlab functions `sinc` and `window` for this. Type `help` followed by the function name in Matlab for more information on the capabilities of the function. You may use filter design tools such as Matlab's `fdatool` if you are familiar with these tools, especially for the trial and error phase, but you need to be able to explain in detail how your filter design works for your report. For this reason, it is recommended to work with low-level functions that you are familiar with.

The pictures below show a valid example design with an $M = 51$ tap linear phase low-pass filter. This filter is designed using the method of windows. The window type is not specified to keep some mystery, but you should naturally in your report disclose the window that you use if you use the method of windows, and also share some thoughts on why you made that particular choice of window.



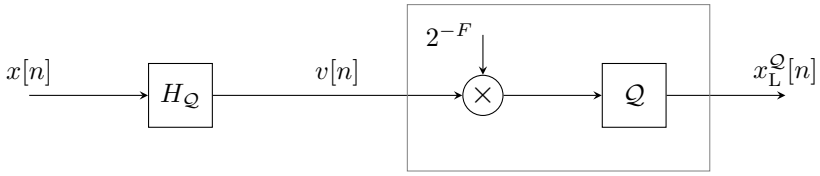
2.2 High-pass filter design

In order to complete the equalizer we also need to construct the high-pass filter, which in this case amounts choosing the delay m of the lower branch.

Taks 3: Give a value for the delay m so that $h_{\mathrm{H}}[n] = \delta[n-m] - h_{\mathrm{L}}[n]$, where the equivalent high-pass filter's impulse response $h_{\mathrm{H}}[n]$ is also a Type I linear phase filter. Argue in this report why this choice of m makes $h_{\mathrm{H}}[n]$ a Type I linear phase filter. Include a plot of the magnitude of $H_{\mathrm{H}}(\nu)$ in the report, in dB scale for $\nu \in [0, \frac{1}{2}]$.

2.3 Fixed point implementation

To be able to implement the filter with integer arithmetics, we will create a filter $h_{\mathcal{Q}}[n] = \big[h_{\mathrm{L}}[n] * 2^F\big]$ for some integer F, where $[\cdot]$ denotes rounding to the closest integer. We will then replace the low-pass filter in the schematic above with the one shown below, where $H_{\mathcal{Q}}$ denotes filtering with the filter with impulse response $h_{\mathcal{Q}}[n]$.

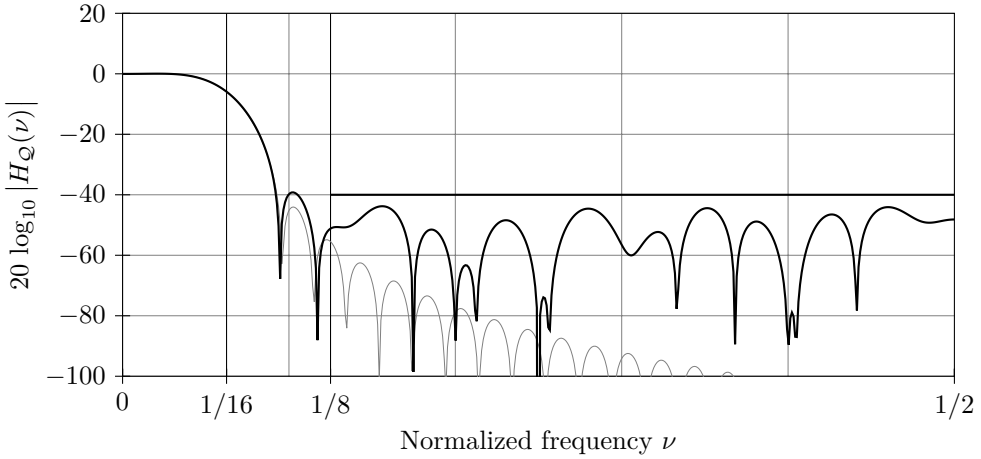


The quantizer $\mathcal{Q}$ quantizes the signal to integer values. If we assume that the input signal $x[n]$ only contains integer values, then filtering by $H_{\mathcal{Q}}$ can be implemented exactly in integer arithmetics. The multiplication with 2^{-F} followed by quantization is easily (approximately) implemented in hardware using an *arithmetic (signed) right shift* by F bits. In C/C++ and many similar programming languages this simply corresponds to `v[n] >> F;` where `v[n]` is a signed integer (`int` or explicitly `int32_t` on the Arduino Due). The implementation above is equivalent to a fixed point application with H_{L} using F fractional bits, appart from the fact that quantization only occurs at one place of the circuit and not for every multiplication by filter coefficients $h_{\mathrm{L}}[n]$ where $n = 0, \ldots, N$ for $N = M - 1$. For the current platform, it is suggested that you keep F below 16 in order not to risk internal overflow in the later implementation.

Taks 4: Compute and plot the magnitude of the frequency response of the equivalent fixed point filter with impulse response given by $h[n] = 2^{-F}\big[h_{\mathrm{L}}[n] * 2^F\big]$. Choose F so that the filter still satisfies the specification given in Task 2, and make sure to specify this value in your report. If

you designed your filter in Task 2 with too tight margins, you may find it necessary to go back and redesign your filter in order to be able to find an integer F that is no larger than 16.

The figure below illustrates a valid solution to based on the example filter of Section 2. Again, the value of F used is left out in order not to bias your own solution, but you should specify your value of F.



The Arduino Due's analog to digital converter (ADC) is a 12-bit ADC. We may thus view $x[n]$ as an integer valued signal in the range $[2^{-11}, 2^{11}]$. Even though the filter $H_{\mathcal{Q}}$ can be implemented exactly, the multiplication with 2^{-F} followed by integer quantization introduces quantization noise.

Taks 5: Assume that $x[n]$ is uniformly distributed in the range $[2^{-11}, 2^{11}]$ and spectrally white. Compute (or approximate) the signal to quantization noise ratio (SQNR) of the fixed point computation of $x_{\mathrm{L}}[n]$, The signal to quantization noise ratio is formally defined as

$$\mathrm{SQNR} = \frac{\mathrm{E}\{x_{\mathrm{L}}^2[n]\}}{\mathrm{E}\{(x_{\mathrm{L}}[n] - x_{\mathrm{L}}^{\mathcal{Q}}[n])^2\}}\,.$$

Give a numerical value for the SQNR in dB ($10\log_{10}\mathrm{SQNR}$) using the filter you previously designed and your chosen value of F, and include an explanation of how you obtained your answer.

3 Epilogue

In the later course lab we will run the filter you designed on the Arduino platform to filter music and sinus-signals, and make some measurements to see how well the implementation is working...

Good luck!