# Lab Assignment, HT 2023
## EQ2300 Digital Signal Processing

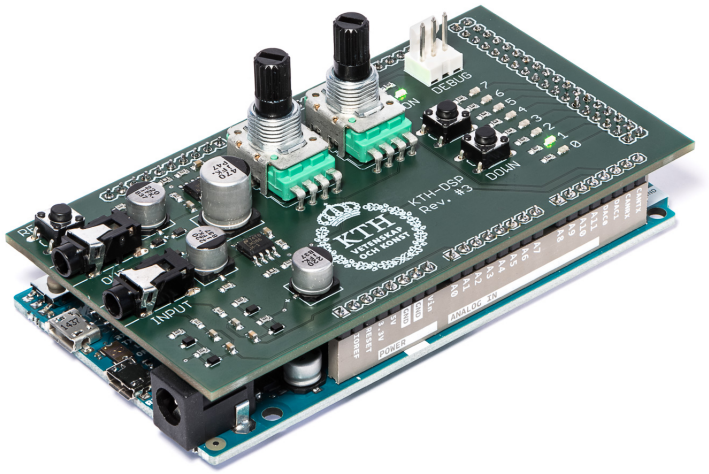


Figure 1: Arduino Due with mounted analog shield for input and output of audio signals

# 1 Introduction

In this lab, we will implement the previously designed filters on the Arduino Due platform shown in Figure 1, in order to complete the audio equalizer from the project. The lab requires close to no preparations, but it is advisable to read the project instructions ahead of the lab to be prepared for what will come.

You are also advised to prepare by installing the Arduino IDE on your own computer, as described in Section 2.2. The lab computers do not have an internet connection, and will not have the files needed to program the Arduino Due, so this needs to be done from your own computer. You can get a head start by completing parts of Section 3.1.1 which involved inserting your filter into the Arduino code. You also need to bring your own music source that can output audio via a 3.5mm audio connector. This can be the computer you have installed the Arduino IDE on, a smartphone (with the appropriate dongle if needed), or a portable music player.

The lab is divided into two sections, Background Information and Lab Tasks. The former provides some information that is good to know in order to be able to complete the lab. The latter is a series of instructions to be followed. The Lab Tasks section also contain 4 checkpoints where you are asked to report your progress to the lab assistant. Once you have successfully cleared these 4 checkpoints, you are done with the lab.

**Important Note:** Many parts of the lab involves listening to sounds and music that has been processed by the platform. As loud sounds can damage your hearing, you should never connect signals to the platform when wearing headphones as you can never be completely sure about the volume.

# 2 Background Information

This section provides some background information on the components of the lab. It is helpful to read through this section once before the lab in order to have a rough idea of what equipment will be used in the lab, and how to use it.

## 2.1 The Arduino Due and the Shield

The DSP platform shown in Figure 1 consist of an Arduino Due and an analog shield to simplify the interfacing with analog signals in the audio range. The Arduino board implements the analog to digital conversion (sampling and quantization), the digital signal processing, and the digital to analog conversion (simple zero order hold pulse amplitude modulation). The analog shield provides some protection for the Arduino, some simple physical inputs, and also analog

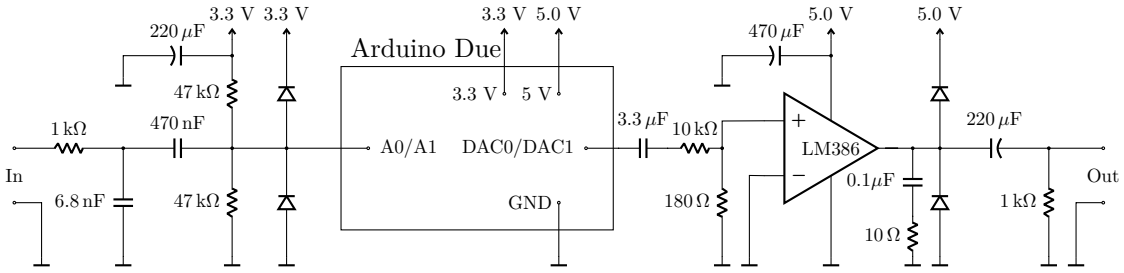


Figure 2: Input and output circuit schematic of the analogue Arduino shield. The schematic illustrate one of the two audio channels.

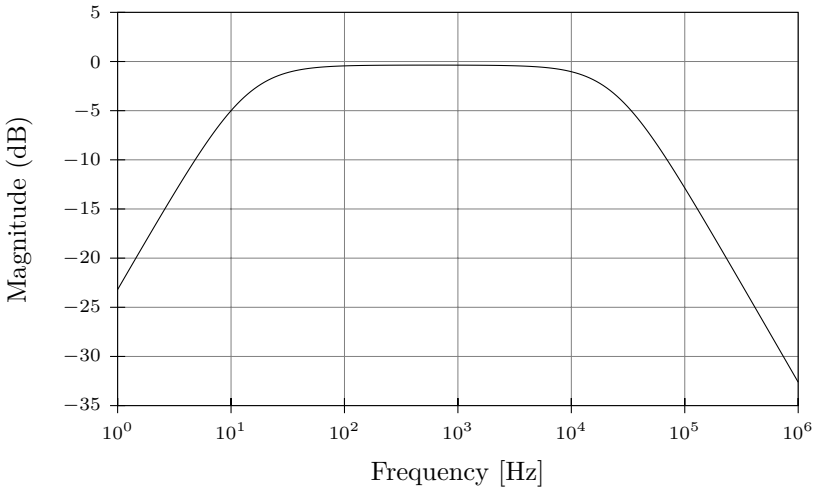


Figure 3: Frequency response for second order analog anti-aliasing input filter.

amplification to enable the direct use of headphones or speakers for listening to the output signals. Audio input can be connected to the board via a standard 3.5mm audio plug (marked input) and audio output can be obtained via a 3.5mm audio plug (marked output). A basic schematic for the input and output analog circuitry is shown in Figure 2. In addition to the audio path, the board has two push buttons for discrete input, two potentiometers for continuous input, and a set of 8 light emitting diods (LEDs). The inputs and outputs can be used to control and show the state of the code that is running on the Arduino.

The input circuit implements an analog bandpass filter with a lower cutoff frequency $f_{\mathrm{L}} \approx 14\,\mathrm{Hz}$, an upper cutoff frequency of $f_{\mathrm{U}} \approx 24\,\mathrm{kHz}$, and a passband gain of 0.96. The transfer function of the filter is shown in Figure 3. The bandpass filter provides a DC separation from the input, and some suppression of higher frequencies that would otherwise be aliased into the audible frequency range by the sampling. This said, the filter is not very sharp and only decays at 20 dB per decade above the upper cutoff frequency. This will potentially still cause some aliasing in the sampling step.

The analog to digital converted (ADC) on the Arduino uses 12 bits, and is closely approximated by an ideal sampler followed by a 12 bit quantizer. The digital to analog conversion (DAC) is closely approximated by a zero order hold pulse amplitude modulator. Figure 4 provides a conceptual illustration for the whole audio chain from input to output, where $H(f)$ denotes the anti-alias filter described before. The software will set the sampling rate to 48 kHz.

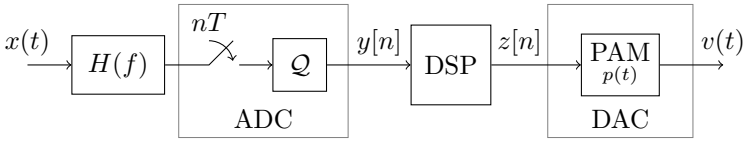


Figure 4: A conceptual overview of the analog-digital-analog chain.

## 2.2 The Arduino IDE

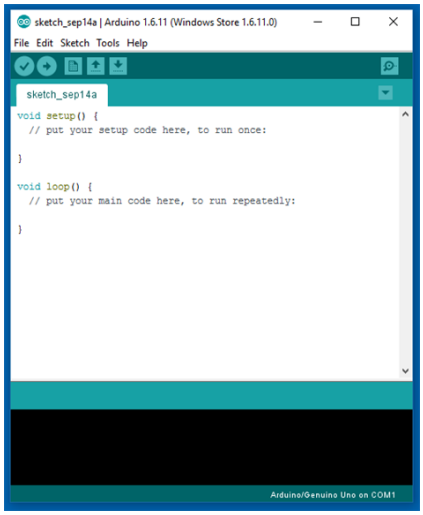


Figure 5: The Arduino integrated development environment (IDE).

Arduinos can be conveniently programmed using the Arduino Integrated Development Environment (IDE), shown in Figure 5. The Arduino IDE works with files called Sketches that contains C/C++ code, and typically a number of calls to Arduino specific library routines. Once configured, it is possible to compile and deploy code to the Arduino platform simply by pressing the upload button at the top of the IDE window.

You need to install the Arduino Integrated Development Environment (IDE) on your own computer to be able to program the Arduino board. Installing and configuring the Arduino IDE can be done by following the steps below.

1. Install the Adruino IDE by visiting `arduino.cc`, download and run the installer for your operating system. The installer is found under the software menu. Choose the local installer, and not the Arduino Web Editor.

2. Start the Arduino IDE, and install the necessary software for the Arduino Due. You can find this under the "Tools → Board → Board Manager..." where you select and install "Arduino SAM Boards (32-bits ARM Cortex-M3)".

3. Now select the Arduino Due (Programming Port) under "Tools → Board" and you are ready to program the Arduino.

Once you have access to a board, and have connected the board to a USB port on the computer, you can easily program the board. The USB cable should at the Arduino end be connected to the programming port, which is the micro-USB port closest to the power adaptor.

To program the board, first select and open a Sketch file (.ino) containing the code you wish to use. If you have selected the Arduino Due board it should just be to press the "upload" button (the little green arrow) to compile and upload the code to the board. In case you get an error message saying "No device found on [Port]", change the used port under "Tools → Port" and try again. If everything is working, you should after some compile time see the code being uploaded and verified. Once this is done the Arduino Due will reset and start running your code.

## 2.3 Visual Analyzer

The lab computers will have a software called Visual Analyzer, shown in Figure 6. The software implements a spectrum analyzer for the signals obtained at the line in port of the computer's sound card. The software is fairly simply to use, and as step by step instructions on how to use it are provided in Section 3.2 it will not be further explained here.

## 2.4 Lab equipment

We will in addition to the Arduino platform and the lab computer also use some of the lab equipment. The most important equipment will be a signal generator, like the PM5130 shown in

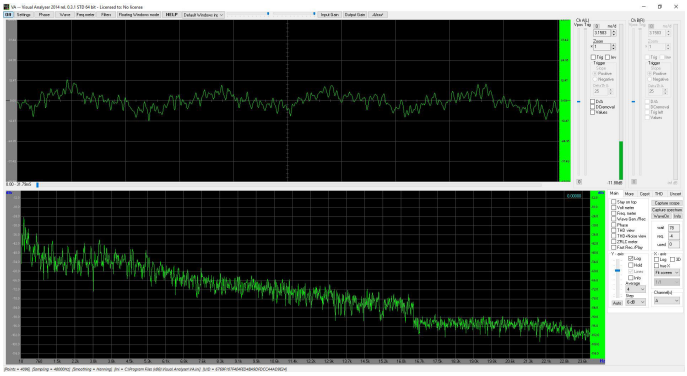

Figure 6: Visual Analyzer 2014 software. A spectrum analyzer for the sound card input.


Figure 7: The PM5139 signal generator.

Figure 7, which will be used to generate sinusoidal (narrow band) signals well above the 20 kHz range covered by standard audio equipment. We will also for the last parts of the lab use an oscilloscope to measure the time that it takes the Arduino to complete the computational tasks given to it. The signal generator and the oscilloscope are fairly self explanatory, or will be known from previous labs. In case you are unsure of how to use any of them, ask the lab assistant. You will need to use an adapter to connect the BNC output of the signal generator to the 3.5mm audio jack of the analog shield.

# 3 Lab Tasks

## 3.1 Setup and testing

### 3.1.1 Getting your filter onto the Arduino

Download the file `EQ2300LabFiles.zip` to the computer where you have the Arduino IDE installed. The file contains two folders, `EqualizerTime` and `EqualizerFreq`, each with a single Arduino Sketch (`.ino`) file. We will begin with the `EqualizerTime.ino` file, which contains a full implementation of the equalizer but with a very simple 3 tap triangular filter that does not meet the design specifications. Open this code in the Arduino IDE.

Browse though the code. In particular, have a look at the function `ADC_Handler` which contain the main code for handling the sampling and reconstruction, as well as the equalizer implementation. The function `ADC_Handler` is an interrupt that is called $48 \times 10^3$ times a second. The code contain a switch statement that selects one out of 4 operations depending on an internal state. The state is shown on the LEDs, and can be changed by pushing the push buttons marked “up” and ”down”. The ADC and DAC writes to and reads from a ring buffer, and by processing the samples in between various effects can be implemented.

In state 0 the code just passes the samples from the input buffer to the output buffer. In state 1 the low pass filter is applied, and in state 2 the high pass filtered signal is computed as described in the project. State 3 applies the full equalizer by weighing together the low pass and high pass filtered streams by the values that are read from the two potentiometers on the board. Pot 1 applies to the low frequencies and Pot 2 applies to the high frequencies.

Edit the `FILTERLENGTH` and `FRACTIONALBITS` definitions in the beginning of the file `EqualizerTime.ino` to match your filter design parameters. Replace the `Filter` array with your filter taps. You can use the Matlab function `export_filter` to export your filter from the project to a text file ready to be copied into the code, so that you do not have to write as much. This Matlab function is located in the zip-file.

### 3.1.2 Testing the functionality of the platform

Connect the Arduino to the programming port, and upload the code to the Arduino by pressing the upload button. If everything works as planned the code will be compiled, uploaded and verified, followed by a reset of the Arduino platform. If you get an error message, ask the assistant for help.

Now you should have a running code on the Arduino. Unless you have disconnected the Arduino it will receive power via the USB-cable. Check to see that the power LED and the LED marked 0 is turned on, indicating that the code is now in state 0 just passing samples from the ADC to the DAC. You can already now test to see that the code is working by connecting a sound source such as a smart phone or a laptop to the 3.5mm input plug, and a set of headphones to the output plug. Check if you can hear any sounds.

## 3.2 Using the Visual Analyzer Software

In order to test the platform in a bit more structured way, we will use the Visual Analyzer software on the lab computers to study the spectrum of the Arduino output. To this end, log on to the lab computer and connect the analog shield output to the line in connector of the lab computer (found at the back of the computer). If the lab computer asks what was connected, answer "line in". Start the Visual Analyzer software (VA64) and configure the software as follows.

1. Click the Settings button to enter the settings menu, and set the sample rate to 48 kHz. If you feel adventurous, you can also change the window function used in the spectrum estimation from Hamming to your favorite window.

2. Set the X-axis to linear by unchecking the Log check-box (in the X-axis box) if checked, see Figure 6 for reference. Keep the Log-scale for the Y-axis.

3. Click the on-button in the upper left hand corner to start the measurements. You should be able to see a background noise spectrum. If you do not, change the scale by sliding the bar in the Y-axis box, and drag the spectrum window up and down.

You can get a more stable spectrum estimate of the power spectrum if you average over several blocks. This setting can be changed by the Average drop-down menu under Y-axis. A value of 4-10 is reasonable. Test that everything is working by connecting a sound source to the Arduino and see if you can see any effect on the spectrum of the output.

If you wish to both study the spectrum and listen to the output at the same time you can either use a Y-splitter (available in the lab) or choose to "listen to the recoding device" by changing the Windows settings and connecting the headphones to the lab computer. You find these settings hidden in under the recording device settings in Windows.

**Checkpoint 1.** *Show the lab assistant that your setup is working and that you can affect the spectrum seen in Visual Analyzer by passing sound though the Arduino platform.*

## 3.3 The aliasing effect

To study the aliasing effect, we will use a narrow band sinus waveform. Generate a 3 kHz siusoid with 0.5 Vpp magnitude, using the signal generator in the lab. Pay attention to the amplitude so that the signals do not clip. Check to see that you can see a single peak that sticks out of the noise floor., and that it appears at 3 kHz in the spectrum plot in Visual Analyzer. Change the frequency of the input signal to see that it also move in the spectrum plot.

Now, increase the frequency of the input signal. What happens when the frequency pass above 24 kHz? Is this consistent with theory? What happens as you further increase the frequency, and at what frequency can you no longer see the input signal in the spectrum? Can you explain what is happening by drawing a connection to Figure 3? What implications does this effect have for the ability of a digital system to operate on analog signals?

### 3.4 Testing the low pass and high pass filters

Change the input signal back to 3 kHz, and set the Arduino platform to state 1 by pushing the up-button. If everything is working your signal should now be passing trough your digital low pass filter. Move the frequency up and down. Does it seem like the sinusoid is being filtered by a filter with a cutoff frequency at 3 kHz, i.e., will it disappear or be suppressed when the frequency is above 3 kHz and left intact when the frequency is below 3 kHz.

Test the same thing with the code in state 2, which implies that the high pass filtered signal is passed to the output. Will the signal be suppressed when it has a frequency below 3 kHz? If not, there may be something wrong with your filter, so ask the lab assistant for help.

**Checkpoint 2.** *Before disconnecting the signal generator, show the lab assistant what you can see when passing the sinusoid though the Arduino in state 0, 1, and 2.*

### 3.5 Filtering music

Test the effects of the DSP code in state 0, 1, and 2 when using a wide band input signal, such as music. Can you see the filter characteristics in the spectrum shown in Visual Analyzer? It is quite possible that you will see a much less clear signal than what is predicted by theory, due to a whole range of disturbances that exists on the hardware platform and in the sound card in the lab computer, but this is ok.

Make sure to also listen to the music in state 0, 1, and 2. Can you hear the difference between the low pass filtered and the high pass filtered signals?

### 3.6 Testing the full equalizer

Now, you are ready to test the fully realized equalizer. Set the code into state 3, feed the platform with music, and move the potentiometers. Can you hear and see the effects that this has at the output?

**Checkpoint 3.** *Show the lab assistant your working equalizer.*

### 3.7 Timing measurements

When implementing real time signal processing applications, it is important that the code is sufficiently fast so that any processing that needs to be completed per sample is indeed completed before the next sample arrives. The Arduino shield has a debug feature that allows us to test this.

The left and right debug pin on the analog shield can be set high (3.3V) or low (0V) by the software. The middle pin is always a reference ground (0V) pin. In the `EqualizerTime` code, the right debug pin is set high at the start of the `ADC_Handler` interrupt, and then set low again when the processing is completed. By measuring for how long the pin is set high before the next cycle, one can directly measure how long the processing takes.

Use this feature, by hooking up the right (signal) and middle (ground) pin to an oscilloscope. Using the autoset feature of the oscilloscope will typically work well to configure the oscilliscope once the signals are connected. If not, the y-scale has to be able to show the range 0 to 3.3V, and the x-scale has to be larger than the $T = 1/48 \times 10^3$ s $\approx 21$ ms cycle time. The trigger level can be set anywhere between 0 and 3.3V.

Use the oscilloscope to approximate the time it takes the equalizer code to compute one output sample? How much of the available time per sample is used by the equalizer implementation, and does it depend on what state the program is in? Can you guess based on your observations what the longest filter that could be used would be?

### 3.8 An overlap save equalizer implementation

Open the code `EqualizerFreq` which contains an overlap save implementation of the equalizer in the frequency domain. In this version, `ADC_Handler` is only used to read from the ADC to the input buffer, write from the output buffer to the DAC, and count samples. Once enough samples have been received for a block, a code that resides in the function `loop` will be executed to do the block processing behind overlap save using the FFT. Please have a look at the code inside `loop` to see if you can recognize the overlap save idea inside the code. The length of the FFT used is 256, although this can easily be changed.

Update the filter in EqualizerFreq so that it uses the filter that you designed.

In `EqualizerFreq`, the right debug pin is now set to go high at the start of the overlap save block, and to go low once the full processing for the block is completed. Similarly to before, measure the proportion of available time that is used for the processing. Is there a difference between the frequency with which the filtering code is called in the time and frequency case? Is the frequency domain implementation more efficient than the time domain implementation? If not, do you have any idea as to why this may be? Is there a benefit to using overlap save on this platform?

**Checkpoint 4.** *Think about the answers to the questions above, and show the lab assistant a working code timing measurement setup, and report the results that you obtained in the timing measurements.*

You are now formally done with the lab although you are free to test the bonus problems below in case you wish to push yourself and test some DSP programming of your own.

# 4 Bonus Tasks

## 4.1 Push the Arduino to the limit

If you can easily design filters using your Matlab code from the project, design progressively longer filters and see how long filters you can use before the Arduino is no longer able to keep up. Does it matter if you use the `EqualizerTime` or `EqualizerFreq` implementation? Which one can handle longer filters. To understand what is going on while doing this, it is helpful to have the oscilloscope timing measurements connected.

## 4.2 Downsampling and upsampling

A downsampling operation by a factor of 2 directly followed by an upsampling by a factor of 2 is equivalent to just setting every odd sample to zero. Add another state (e.g., state 4) to the code in `EqualizerTime.ino`, where every second sample is passed as is to the output and every second sample is replaced by a zero. Test the code with a sinusoidal input set to somewhere between 3 kHz to 6 kHz. What do you see in the output spectrum? Is this consistent with what is predicted by theory?

## 4.3 Linear interpolation

Modify the previous code so that instead of replacing every second sample with a zero, you replace it with the average between two neighboring samples, i.e., apply linear interpolation. Repeat the experiment by passing a sinus signal through the platform and see what the output spectrum looks like. Can you explain this using the theory from the course?

## 4.4 Report back to the lab assistant and the teacher

In case you have completed any of the bonus problems, please let the lab assistant know. This will be valuable information to have as we further develop the project and the lab for coming years. In case you have an idea of what else could be done with the platform, we would be interested in knowing this too.