

ELEC 3004/7312: Signals Systems & Controls

## Prac/Lab 1: Introduction to the Arduino Due + Sampling & Reconstruction on the Arduino Due

Revised March 14, 2016

### Pre-Lab

The Pre-laboratory exercise for this laboratory is to:

- Revise basic sampling theory
- Read 1-2 pages about the Arduino Due board – enough to get some idea about it
- Read about analog low-pass filters.

### Laboratory Completion

**Доверяй, но проверяй**  
(doveryai, no proveryai – “*Trust, but verify*”)

#### Laboratory Completion:

Please work **together** on the lab in groups of 2-3.

Treat the questions at the end of the Experiments as thought questions – questions that you should be able to answer, but not that you have to answer explicitly, although you could be asked these during a tutor group review. Remember the concepts in the laboratory might be assessable in future problem sets and/or exams.

#### Tutor Group Review:

At various stages in the lab (or at the end) the tutors will come around to check progress. They **may** ask some questions to check your understanding. Each person in the group **may** get asked different questions at the tutor’s discretion.

The tutors are just checking that you understand the core ideas of each experiment and its parts.

#### Extra Credit Points:

Unlike previous years, this practical laboratory is worth 0 Extra Credit Points on the final exam. References to extra credit are remnants from previous offerings and should be kindly ignored please.

## Experiment I: Introduction to Arduino Due + Sampling and reconstruction

The Arduino Due (Doo – ay) is an open-source development platform which is suitable for embedded development and scientific experimentation. The Arduino family consists of devices as small as a 20 cent coin, through to extended boards with many I/O ports. Our device uses an 84 MHz 32-bit ARM microcontroller with plenty of I/O including Analog inputs and outputs in the form of ADC, PWM and DAC. There are also many digital I/O.

This lab is to get you familiar and comfortable with using the software and connections on the board. This particular board is designed for a 3.3 Volt interface, so we have to be careful with what signals are attached to it.

The ADC, PWM and DAC have raw unfiltered connections, but we can attach external filter boards to smooth or limit signals. An important part of sampling is the limit caused by the Nyquist frequency, so we will explore that in this set of experiments. In any system for experimentation, we need to establish base values so we can observe actual effects of the system, so we will begin with a simple output calibration, then make some changes to observe the effects on the system. After that we use a simple input signal to check calibration and establish inherent system limits, so we can be sure that the experimental results are due to rational causes.

We will start with the Look Up Table, where a known signal has been digitised and is pushed out to the DAC for observation. We can change the contents of the LUT for a different signal, or we can manipulate characteristics of our control system to change the output.

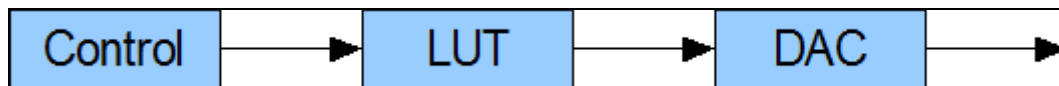


Figure 1. Block diagram of Look Up Table system

The fundamental part of any digital control system is the ability to sample, process, then re/construct a signal. This is basically the process of analog-to-digital conversion (ADC or A2D), Digital Signal Processing (DSP), then digital-to-analog conversion (DAC or D2A), although some systems may operate only in the digital domain internally once an external signal has been converted and processed.

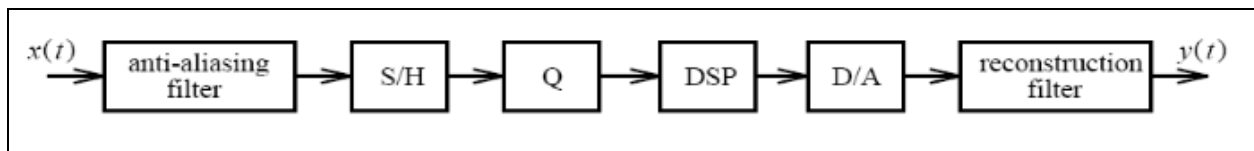


Figure 2. A block diagram of a practical DSP system

Figure 1 shows a block diagram of a simple lookup table signal generator, while Figure 2 shows a typical digital signal processing (DSP) system. On the Arduino Due, the ADC has no low pass (anti-aliasing) filter, however additional 10 kHz lowpass filters can be inserted at input and output. The Due codec is implemented as separate analogue to digital (A2D) and digital to analogue (D2A) converters, these being represented by the sample and hold (S/H) plus quantiser (Q) and D/A blocks in Figure 2 respectively. Finally, the DSP block in Figure 2 is where the equations for implementing the desired digital filters are performed.

### Equipment

1. Arduino Due 32-bit development board + USB cable/s
2. PMOD-CON4 RCA board/s (optional)
3. Oscilloscope with function generator
4. 2 x RCA male to BNC male cable approx 0.5 - 1 m
5. 1 x E36 DAC Filter board and 1 x E36 ADC Filter board (UQ designed) optional
6. RCA-3.5mm adaptors + powered speakers + 3.5mm stereo to 2 RCA male Y-cable
7. BNC male to BNC male approx. 0.5-1m, RCA-BNC adaptor, and T-piece F-M-F

## Preparation

Read the following documents to get an idea of the system we will implement:

1. Arduino Due

<http://arduino.cc/en/Guide/ArduinoDue>

2. Analog low pass filter

[http://en.wikipedia.org/wiki/Low-pass\\_filter](http://en.wikipedia.org/wiki/Low-pass_filter)

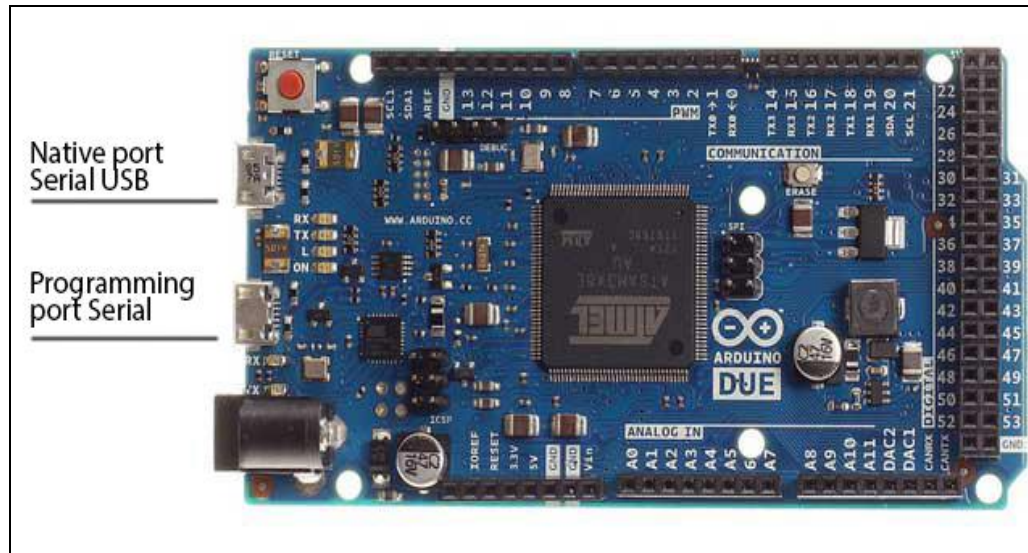


Figure 4 Arduino Due

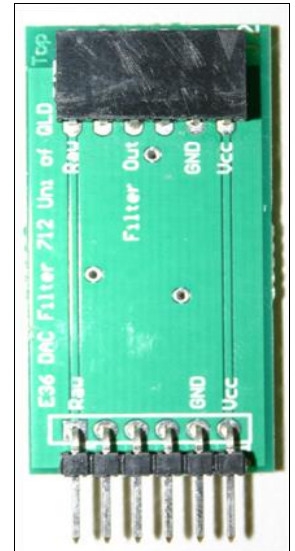


Figure 3 DAC Filter board

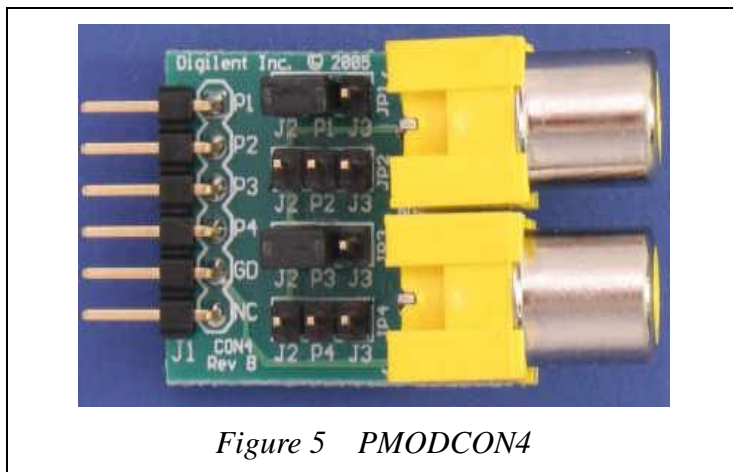


Figure 5 PMODCON4



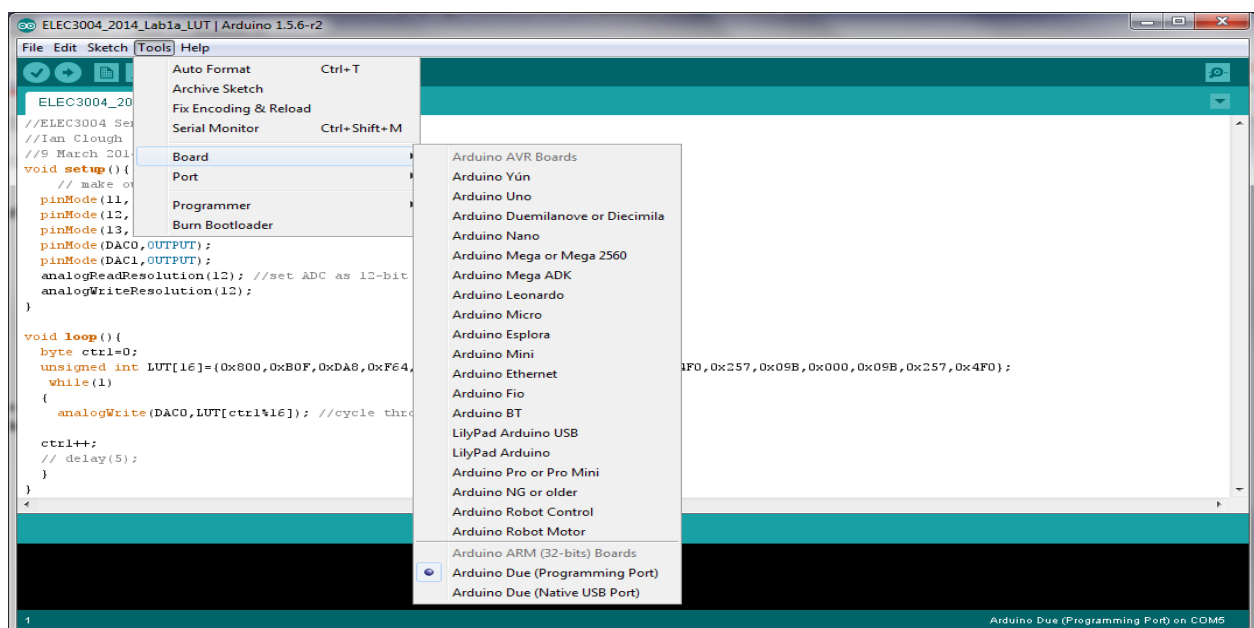
Figure 6 USB MicroB cable

We will setup the software (IDE) and basic connections to the Due, then run our sanity checks on the hardware and connections.

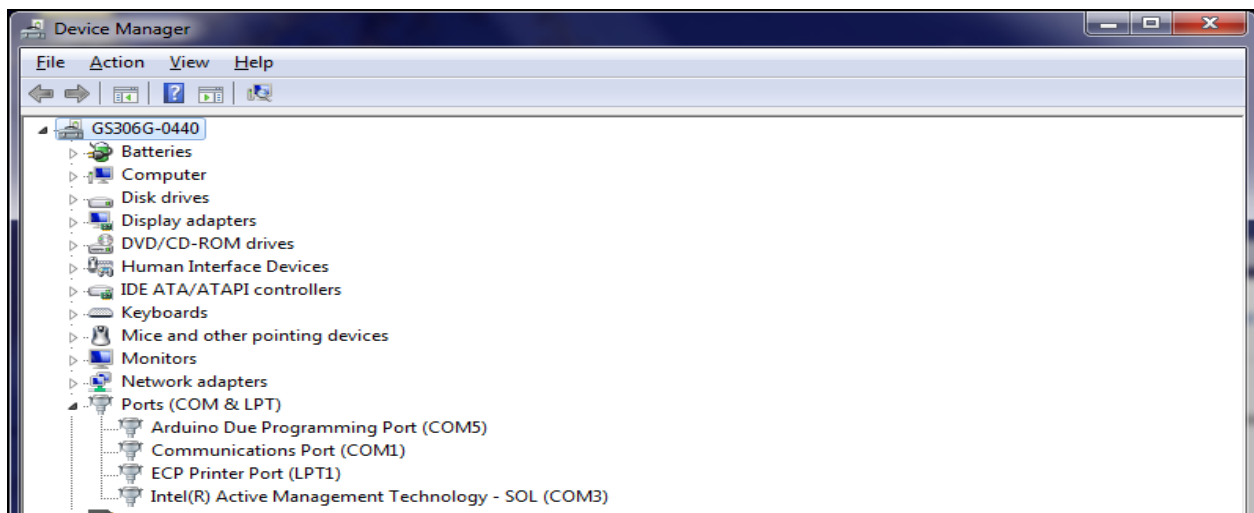
Procedure: Each part will take about 20 - 30 mins

## Lab 1a: Getting familiar with the Arduino Due by generating a sine wave

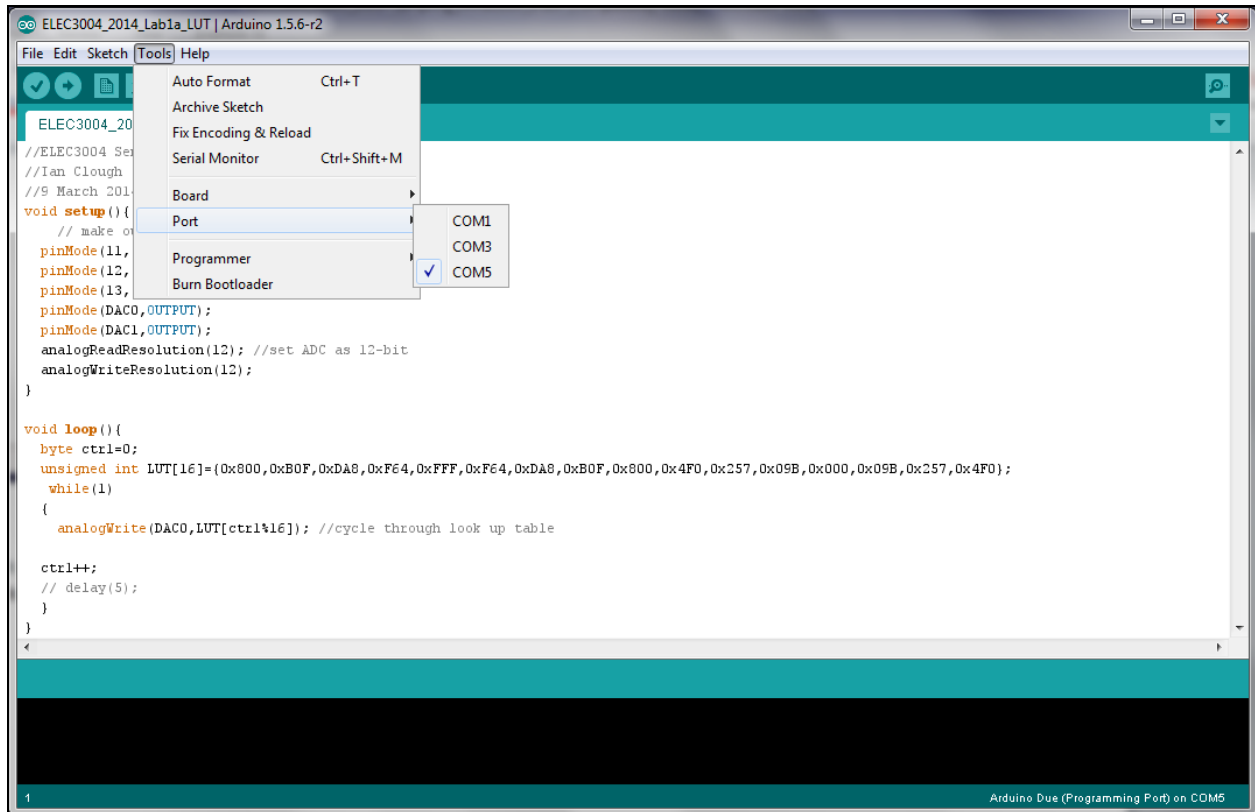
1. Start the Arduino 1.5.6-r2 IDE using the desktop icon, or navigate to where it exists in Program Files.
2. Open the ELEC3004\_2014\_Lab1a\_LUT file and look at the code.
3. Connect the USB to Micro-B USB cable between the PC and the Due on the Programming Port as shown in Figure 4. Windows may give you a message about needing to load a driver. If so let this complete.
4. You will need to setup the IDE for the Due on the Programming port by selecting Tools – Board – Arduino Due (Programming Port)



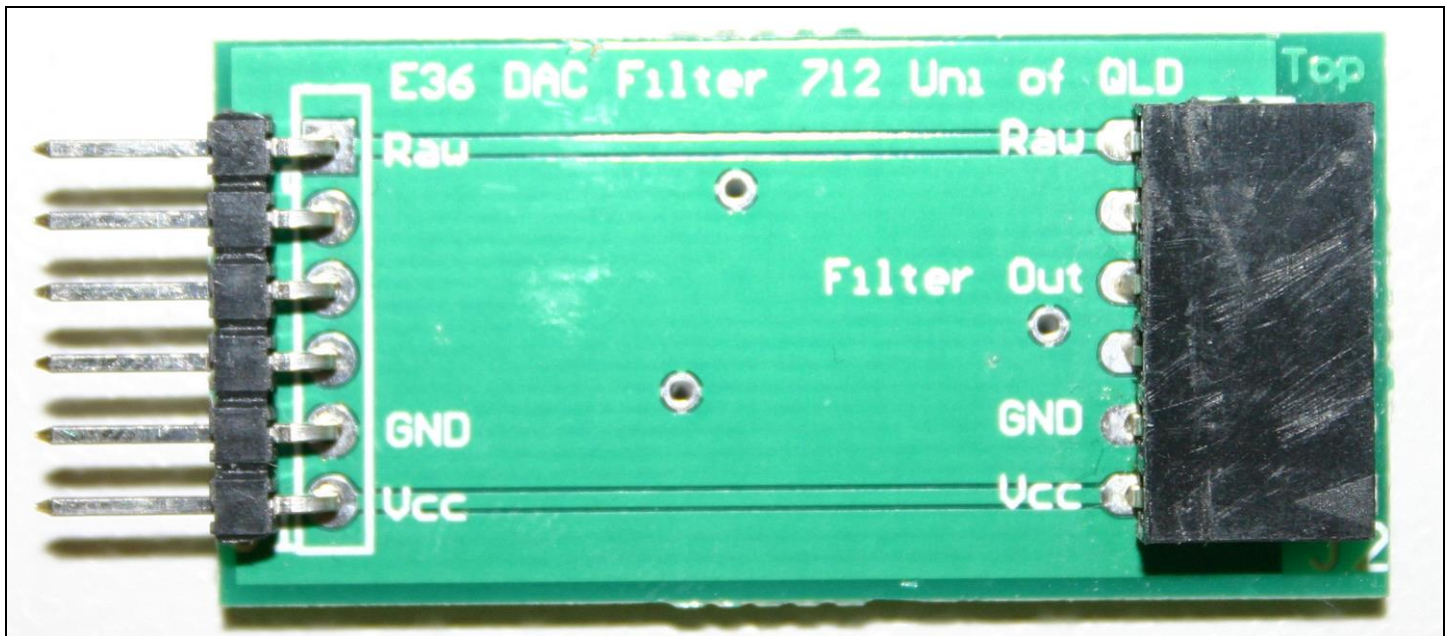
5. Start Device Manager by typing in the **Start Search** box, or in the **Run** box, the following command: `mmc devmgmt.msc` You should now see which COM port the Due is connected to.



6. Now set the COM port to what is shown in Device Manager.



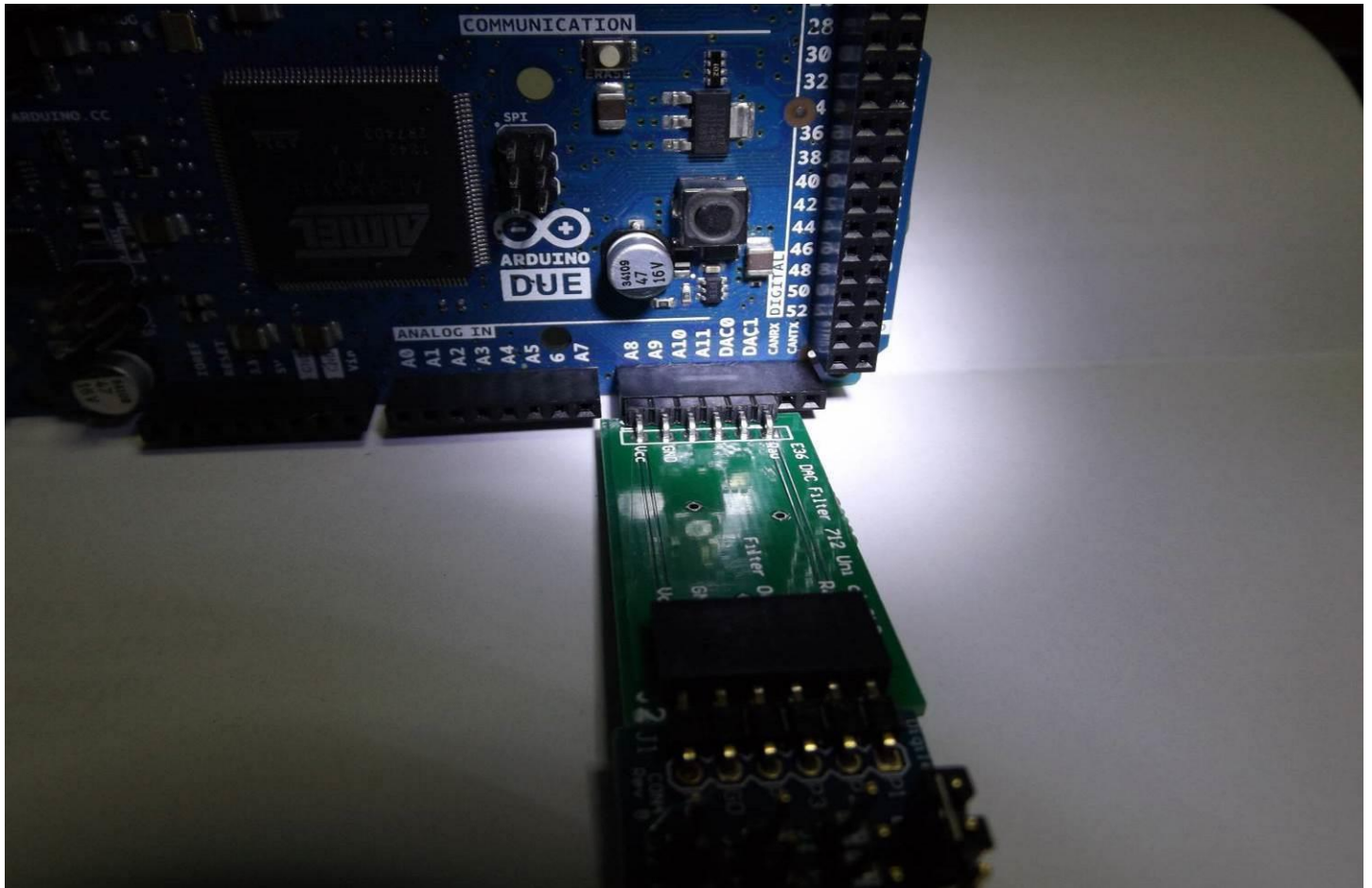
We need to connect the output signal of the Due to an oscilloscope. This can be done directly or via filter board.



*Figure 7 DAC Filter board*

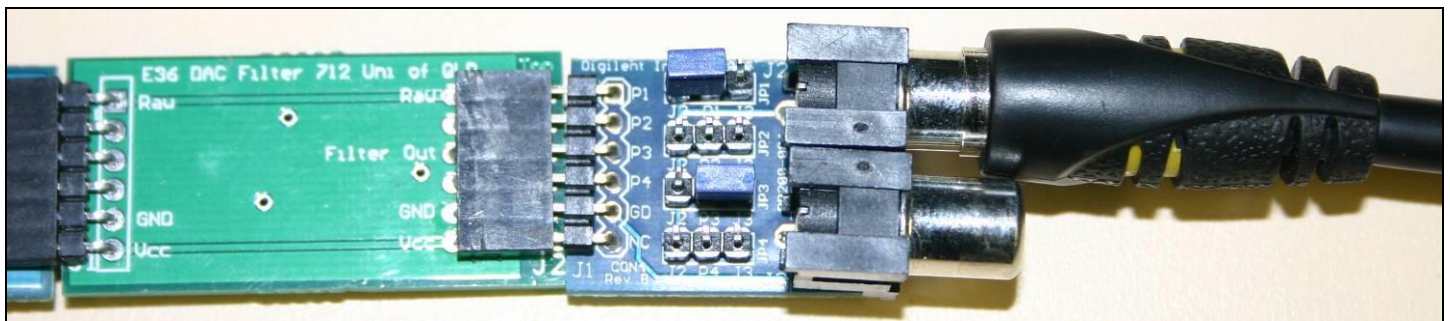
You can see that the filter board has a Raw input on the left side, as well as a Raw output and Filter Out on the right side. The Raw input side will be connected to DAC1 on the Due.





*Figure 8 Due Board with DAC Filter attached*

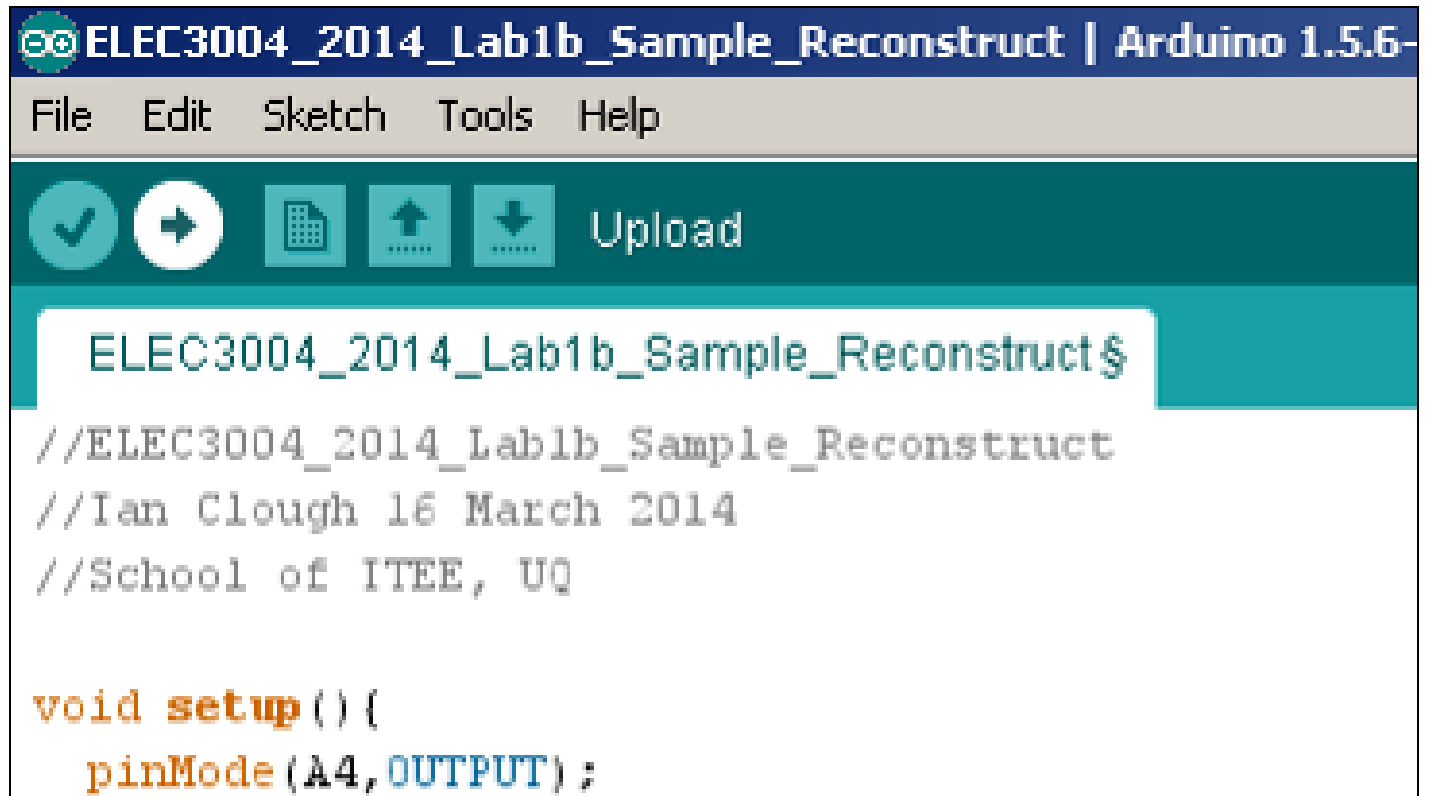
Here we have fitted the DAC Filter board so that the Raw input of the filter is attached to DAC1. GND and VCC for the filter board are provided by setting the ports for A8 and A9 as OUTPUTs with a LOW or HIGH accordingly – see the code for Lab1a.



*Figure 9 DAC Filter with PMODCON4 and RCA cable which goes to oscilloscope*

Figure 9 shows a typical connection. The other end of the RCA cable is connected to either CH1 or CH2 on the scope. This arrangement shows the Raw output. Connect an RCA cable in the lower socket if you want the Filtered output

**If you want to see Raw and Filtered output simultaneously, connect an RCA/BNC cable between the PMOD-CON4 upper socket and CH 1 on the oscilloscope, then connect a second cable from the lower PMOD-CON4 socket to CH 2 on the oscilloscope.**



The screenshot shows the Arduino IDE interface. At the top, the title bar reads "ELEC3004\_2014\_Lab1b\_Sample\_Reconstruct | Arduino 1.5.6-". Below the title bar is a menu bar with "File", "Edit", "Sketch", "Tools", and "Help". Underneath the menu bar is a toolbar with icons for a checkmark, a right arrow, a document, an upload arrow, and a download arrow, followed by the "Upload" button. The main code window displays the following code:

```
ELEC3004_2014_Lab1b_Sample_Reconstruct$
//ELEC3004_2014_Lab1b_Sample_Reconstruct
//Ian Clough 16 March 2014
//School of ITEE, UQ

void setup(){
  pinMode(A4, OUTPUT);
```

Click on the Upload button. The IDE will compile the program, and if there are no errors, it will upload the program to your board, where it will immediately start running. This can take up to a minute.

You should now have a sinewave being sent to DAC1. You may look at this with your oscilloscope to verify that it is working correctly.

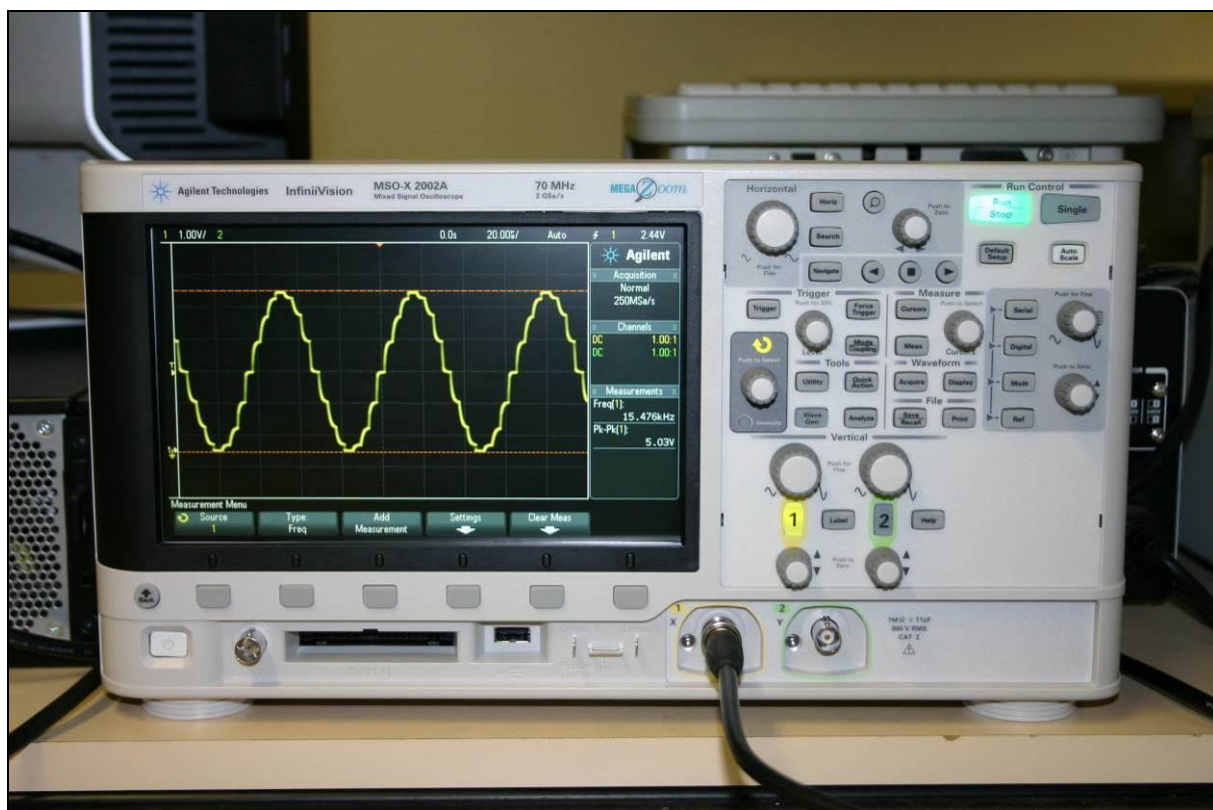
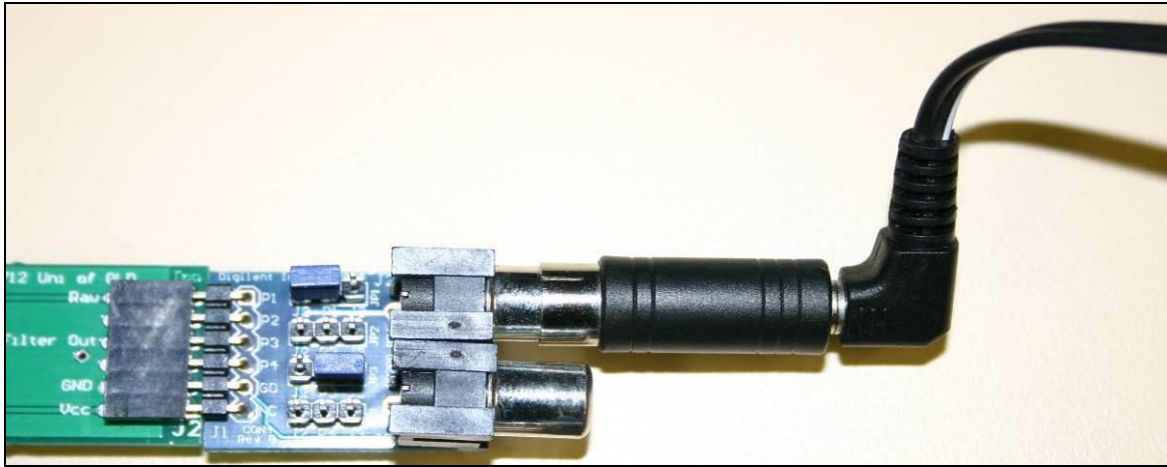


Figure 10 A typical DAC output. Your amplitude and frequency may differ.

Plot the output waveform/s on the oscilloscope. The filter boards use a 2<sup>nd</sup> order Sallen-Key lowpass filter with corner frequency ( $F_c$ ) of 10 kHz.

What is the amplitude and frequency of the sine wave generated **on each available channel**?

Suggest how you might modify the sinewave code so that a LUT with only 8 points is required. What are the advantages and disadvantages of this approach?



*Figure 11 Using RCA to 3.5mm adaptor with speaker cable*

If you want to listen to the audio output from any experiment, you can use the RCA to 3.5mm adaptor and connect it to headphones or the powered speakers. Some frequencies will be inaudible, others very obvious, so adjust the volume on the speakers for a safe listening environment.

Try connecting to the filtered output and comparing it to the raw output. What differences do you notice?

Experiment with sampling rate by changing the delay value in the line of code which says:

**delayMicroseconds(0); // You can adjust the replay rate by altering this value**

See what effect it has on the filtered output.

## Lab 1b: Sampling and Reconstruction (approx. 20 mins)

Initially we will read a value from the ADC then push it straight back out to the DAC. This is a system sanity check. By doing the previous experiment, we have tested whether the DAC works correctly. Now we can check the ADC with a simple loopback program.

Variations on the system allow us to change the sample rate or bit resolution, do some DSP like multiplication or offset shifting, or some type of digital filtering, but digital filtering is for a later lab, so we won't discuss it further here.

```
while(1)
{
    unsigned short X0 = analogRead(A5); // get a sample from Raw Analog input 5
    delayMicroseconds(0); // delay between read and write

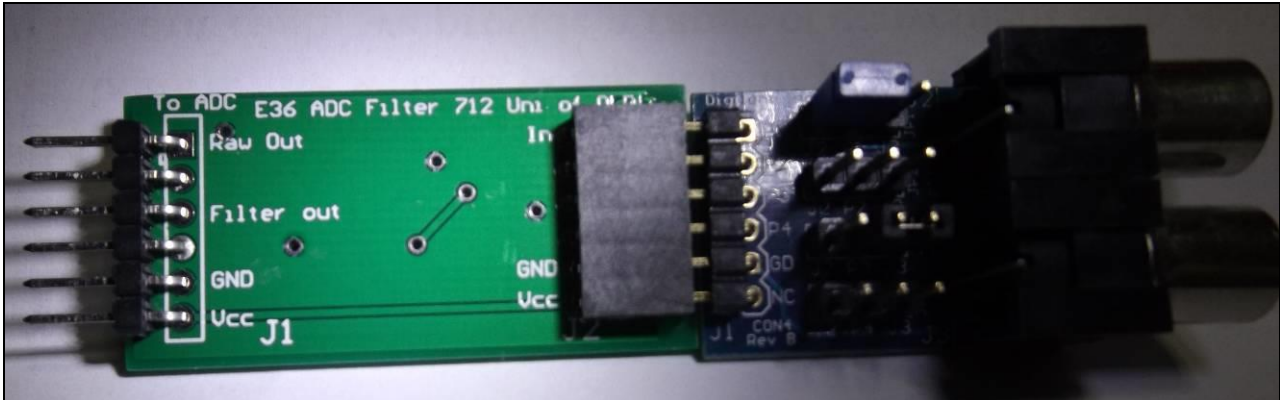
    // Your DSP routines, if any, go in here
    // you could try multiplying or dividing the input signal, or try adding or subtracting an offset

    analogWrite(DAC1,X0); //write the result to DAC1
    delayMicroseconds(0); //optional delay between write and read
} // end while
```



Using the Wave Gen on the oscilloscope, you can supply a signal such as a sinewave to the input of the ADC. Alternatively, we could use a sound source such as the PC headphone output, or a music player or smartphone or tablet etc. Because these have bipolar outputs and the ADC input is Unipolar, we need some kind of offset to control the signal, otherwise we would observe clipping in the output.

We can do this easily with the ADC Filter board, which has circuitry to handle bipolar input signals. See the circuit schematics in the appendix for further details.

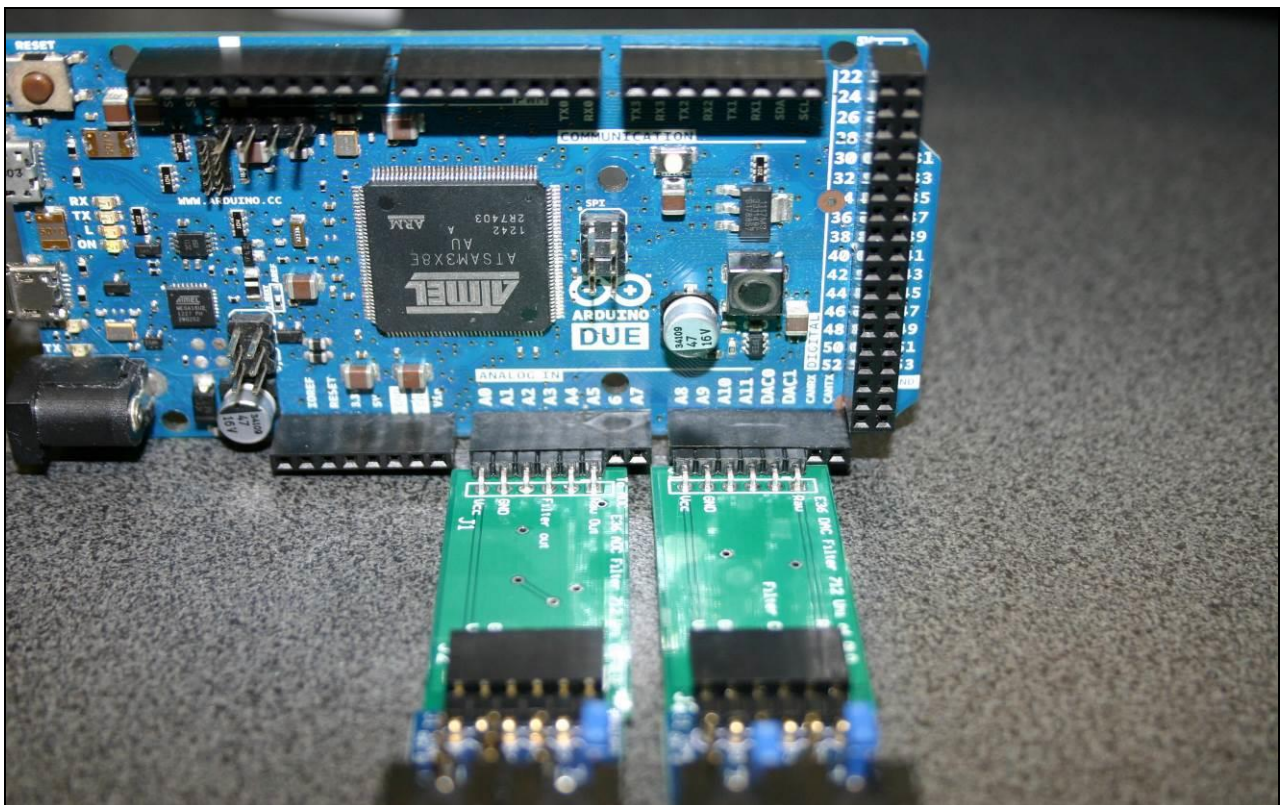


*Figure 12 ADC Filter and PMODCON4*

In Figure 12 you can see one input on the right-hand side of the filter board, and 2 outputs on the left side.

The Raw Out can be read on one of the Due's analog input channels, Filter Out on another. Modifying the supplied code will allow this, such as changing the default channel

**unsigned short X0 = analogRead(A5); // get a sample from Raw Analog input 5**



*Figure 13 ADC and DAC Filter boards fitted*  
**Fit both filter boards before proceeding.**

To find the fastest sampling and reconstruction rate, attach the output of the Wave Gen to the top RCA connector on the PMODCON4. Ideally, you should set the offset to 1.65 V. Make sure the maximum signal value doesn't exceed 3.3 V pp. Choose a sinewave at 1kHz and check the DAC output. It should also be 1 kHz and about 2.2 Vpp. Now increase the frequency of the wave gen until the output of the DAC goes to almost zero. This will be your approximate sample rate. Why? **Hint – Nyquist and sampling theory. What happens if you multiply a sinusoidal signal by itself?**

- Now keep increasing the frequency and observe what happens to the output signal.
- Reduce the frequency back to about 100Hz and slowly increase the frequency until the raw signal starts to look like a staircase. What does the filtered version look like?
- Modify the code `delayMicroseconds(0); // delay between read and write` to decrease the sample processing rate to about 10 kHz.
- Start the input frequency at about 100 Hz and increase it like before. Now when does the output frequency of the raw output go to nearly DC? How about the filtered output?

**You can see the input signal as well as the output signal by using a BNC T-piece F-M-F. Connect a BNC male to male cable from the output of the wave gen to the T-piece placed on Channel 1 input. Now connect a BNC-male to RCA-male cable from the T-piece to the PMODCON4 as shown above in Figure 12.**

You can experiment with changing the amplitude or offset of the signal written to the DAC in the line of code:

**`analogWrite(DAC1,X0); //write the result to DAC1`**

using something like `analogWrite(DAC1, X0/2)` or `2*X0`, or `X0+500` or similar. If the value of X0 after a multiplication is large enough, you will get overflow. What does this look like in theory and what does it like in the output of this system?

### **Filtered input:**

You can repeat the process using the Filtered input by using analog input A3 instead of A5. What do you think will happen to input signals over 10kHz?

### **Saturation:**

The Due board can tolerate input values just slightly outside the range of 0 to 3.3 V. However, the ADC Filter board will protect the Due against values up to about 5Vpp.

Set the wave generator /function generator back to 1kHz sine wave, offset of 1.65V and signal level of 3Vpp. Now slowly increase the signal level beyond 3.3Vpp towards 5Vpp and compare the output signal against the input signal

## **Lab1c Echo. (About 20 mins)**

This experiment is a bit of fun to get used to DSP dealing with memory elements in the Due, which has about 96 kilobytes of SRAM. The program variables use about 6K, leaving approx. 90 K. As we are using 12-bit data, we have to use 16-bits to store it, so we actually have about 45000 16-bit memory elements available, but this is plenty of memory to explore some great musical effects.

Your audio input can come from the function generator or any other audio source with a headphone socket, but it will need to be connected through the ADC Filter board to provide the offset to convert a bipolar audio signal

into a unipolar ADC signal.

You can control the effect of the echo through a number of variables. The echo can be expressed in a simple equation,  $Y(n) = X(n) + \alpha X(n-D)$ , where  $(n)$  refers to the current sample,  $\alpha$  refers to the relative amplitude of the delayed signal  $X(n-D)$  and varies from 0 to about 1, and  $D$  refers to the delay or echo time, expressed as a number of samples.

If you know the sample rate, you can calculate how many samples are required for a delay of 1 second. Alternatively, you can calculate the delay time if you know the sample rate and the number of memory elements available. In our case the maximum is 45000, so a sample rate of 90 KHz would be 0.5 seconds, and 20kHz would be 2.25 seconds. However, we don't need to use all of that memory space to achieve a useful effect. If we arrange the memory as a circular memory, we can keep writing into forever, but we will regularly overwrite existing samples, but this is not a problem if we have a constantly changing input, such as audio.

If you want to hear only the current sample, then make  $\alpha = 0$ , although this is not a very effective echo device, so choose some non-zero value. You can experiment with the value of  $D$ .

### ***Procedure:***

Open the relevant file, then upload to the board. Make sure you have both filter boards fitted and use the RCA to 3.5mm adaptor to connect the output to the powered speakers.

Experiment with values for  $\alpha$  and  $D$  while playing various audio inputs. Try some of the short samples available on the ELEC3004 website, or try the excellent sounds available from <http://www.soundjay.com/button-sounds-1.html>. You can also use the Wave Gen and change the frequency. Try changing it slowly and quickly. Also change the sample rate to see the effect on the echo.

## **Lab 1d: Echo with feedback**

In this experiment you will modify the code from Lab 1c to get feedback in the echo. This will require storing the output instead of the input, so you could theoretically have infinite echoes.

Common examples of echo with feedback are reverb <http://en.wikipedia.org/wiki/Reverberation>, and comb filters [http://en.wikipedia.org/wiki/Comb\\_filter](http://en.wikipedia.org/wiki/Comb_filter). Both of these can be used for musical effects, or for simulating the reflective characteristics of acoustic environments such as amphitheatres, canyons, tunnels and hard surfaces.

Change the code so that you are storing the output signal, rather than the input signal. Once again you can modify  $\alpha$  and  $D$ , as well as the sample rate. If you set  $\alpha$  to a fairly high percentage, you should get a signal which decays very slowly. What would happen if  $\alpha$  was to set to 1.0 or higher? Why not try it with a small amplitude signal and see using values of 0.9, 0.99, 1.0, 1.01 and 1.10.

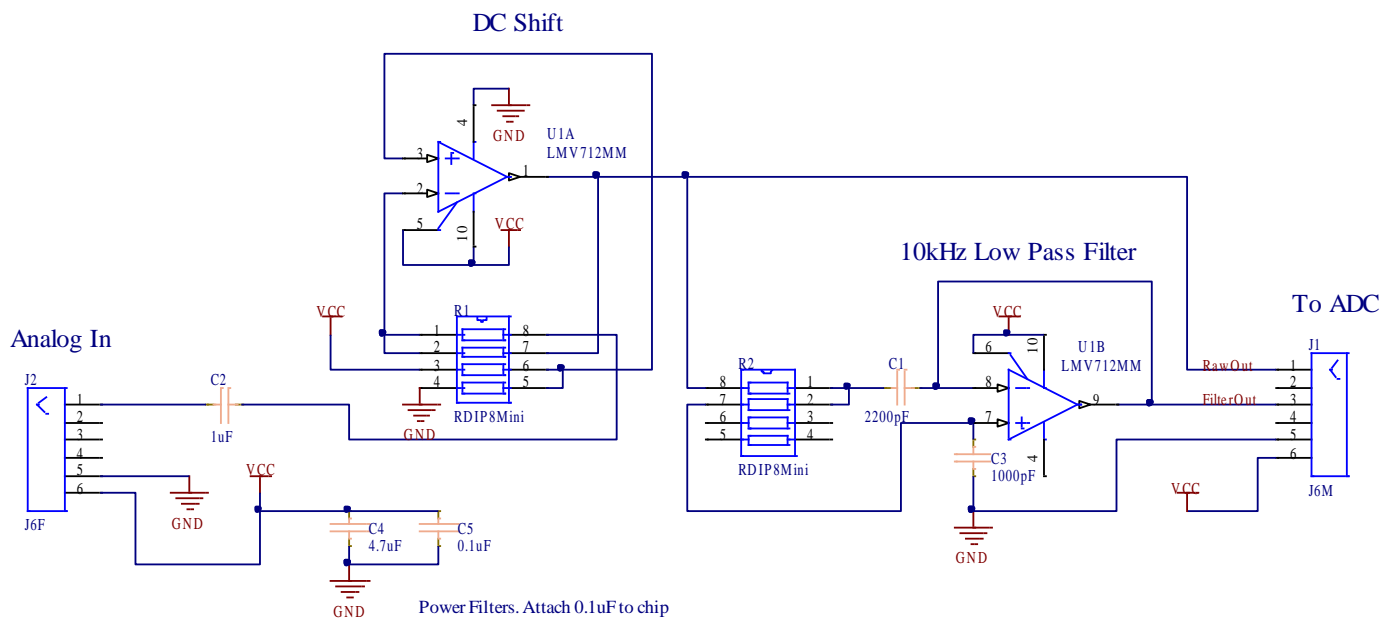
Try one of the short button sounds from *Soundjay.com* with various values of  $D$  and sample rate.

**END OF LAB1**

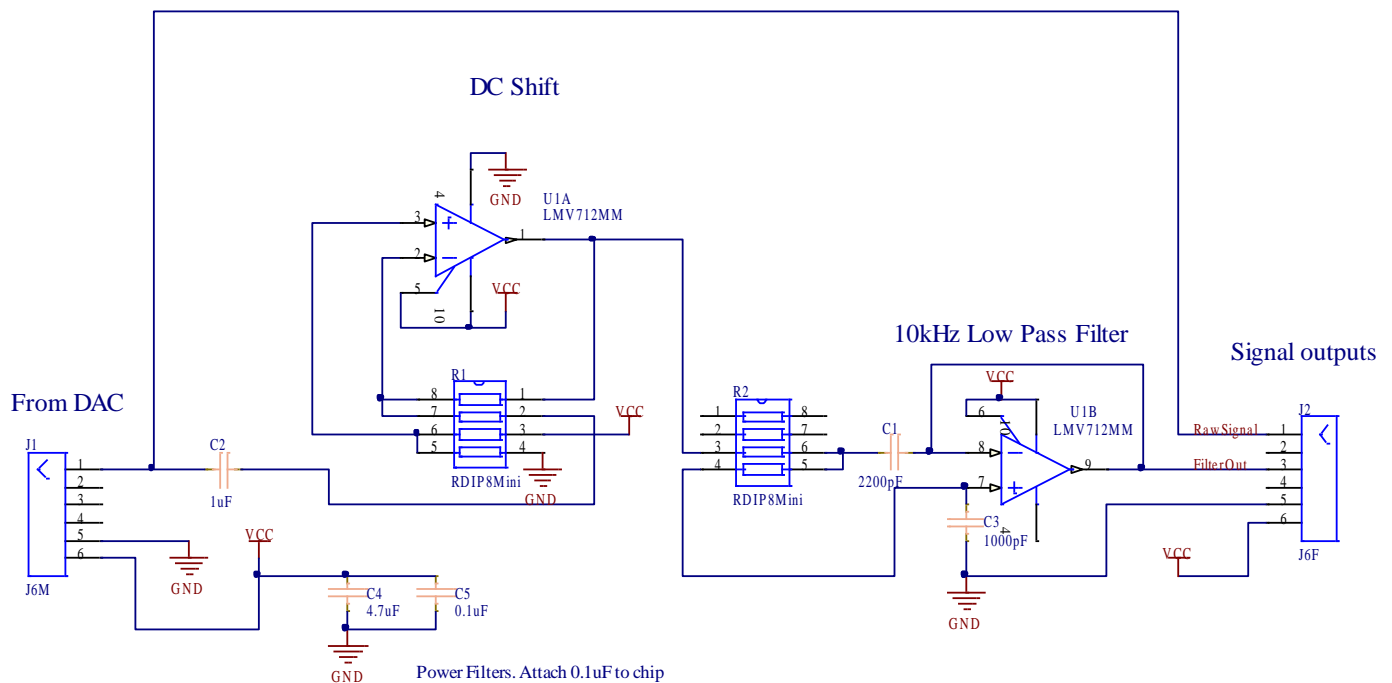
## Appendix:

### Schematic diagrams for filter boards:

Resistors are 10k.



*Figure 14 ADC lowpass filter*



*Figure 15 DAC lowpass filter*



**Experiment 1:*****FYI*****Introduction to the Arduino Due + Sampling & Reconstruction**

Name: \_\_\_\_\_

Student ID: \_\_\_\_\_

Date: \_\_\_\_\_

Group Name/Members: \_\_\_\_\_

\_\_\_\_\_

Section Completed	Summary Comments
Complete Lab 1a	
Complete Lab 1b	
Complete Lab 1c	
Complete Lab 1d	

*This page is intentionally blank and may be used as scratch paper*