Lab 5:
Analog-to-Digital and Digital-to-Analog Conversion

As you already know, signals produced by natural sources are inherently analog (e.g., sound or light signals), whereas most of the infrastructure in use today to analyze, store and transmit signals is digital. Thus, digital-to-analog conversion (DAC) and analog-to-digital conversion (ADC) are necessary in order for many modern electrical devices to interact with the natural world. For example, a CD player has to convert the digital information on the disc into an analog sound signal so that you can hear it, and a digital camera has to convert analog light intensities into a set of digital values to store a "picture". An important part of the conversion process is understanding the issues and trade-offs associated with sampling an analog signal and then quantizing the signal (rounding off the amplitude of the signal to allowed digital levels, as you explored in the pre-lab questions). The first part of this lab uses a data acquisition (DAQ) board and a LabVIEW program to digitize analog signals for us, and allow us to understand the effects of sampling rate and quantization on the conversion results. In the second part of the lab you will use simple DAC to convert Arduino's digital outputs into analog signals.

1. Objectives:

1. To understand the uses of ADC and DAC, and to simulate them in software using a computer data acquisition board;
2. To build D/A converter using resistors only, and use it to convert Arduino's digital outputs into analog signals;

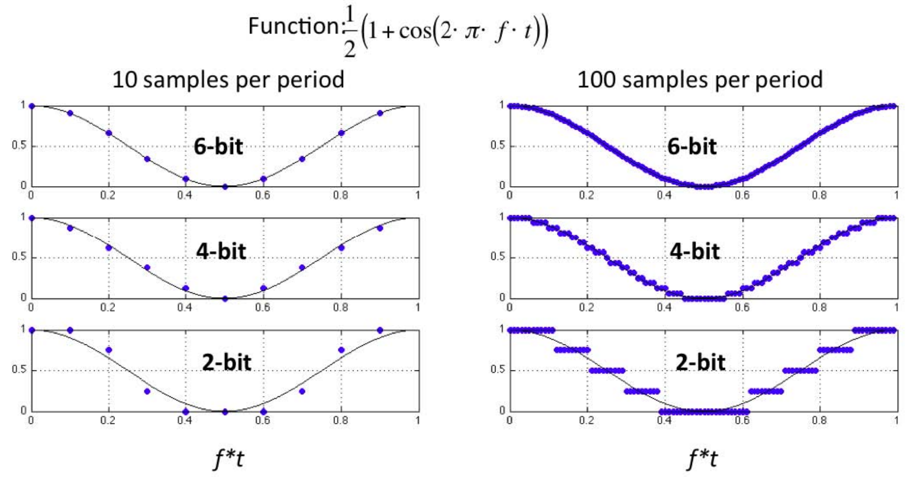

Figure 1: An analog signal (solid black line) sampled with different number of samples, and quantized with different number of bits. Blue dots represent sampled and quantized signal.

2. Pre-lab Questions:

Figure 1 illustrates how an analog signal might be both sampled and quantized. As you know, the more samples we have, and the more bits we use to represent the quantized value, the closer to the original signal we get.

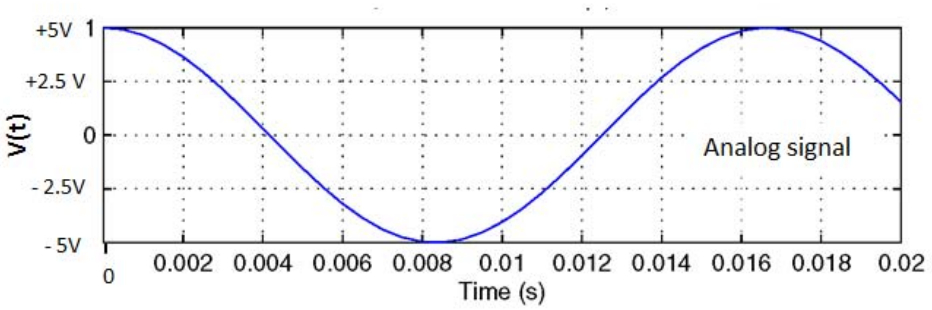

Figure 2: An analog signal that we want to digitize

Now consider an analog time-varying voltage signal shown in Figure 2, that can be described by the equation $V(t) = 5\ cos(2\pi f t)\ Volts$, and where $f = 60\ Hz$.

a. Suppose that your sampling rate is $4 \times 60\ = 240\ Hz$. On Figure 2, indicate where the samples would be taken (i.e. draw a circle at the position of each sample).

b. If your sampling frequency were increased to 480 Hz, what might the sampled values look like? Again indicate the samples in Figure 2, using different symbol.

c. In addition to sampling the signal, we need to quantize the signal. What can you say about the impact of large number of bits used for quantization on the quality of the digitized signal?

3. DAC using National Instruments DAQ Board and LabView Program

The DAQ board (shown in Figure 3) connects to the computer through the PCI or USB bus depending on which version of the board you have on your bench. (**Note:** The USB version can fit in your hand while the PCI version is considerably larger!) The boards used in the ES50 lab can digitize analog signals with a maximum resolution of 16bits and a maximum sampling rate of 10kHz. The quantization range for the DAQ boards can be varied, but in this lab it will be in [-5 V,+5 V] range.

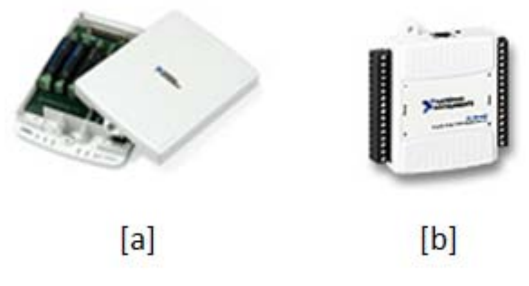

Figure 3: (a) large and (b) small DAQ boards

The DAQ board receives signals through a cable and interface box. For this lab, we are interested in only two pins on the interface boxes:

- **PCI version**: **pin 68**, "Analog Input Channel 0", and pin **34**, "Analog Ground" will be used. In addition, there should be a wire (jumper) connecting pin 34 with pin 24.
- **USB version**, **pin 2**, "Analog Input Channel 0", and pin **1**, "Analog Ground" will be used. Also, there should be a jumper connecting pin 1 and pin 3.

Before continuing, make sure that the cable and your interface box are properly connected, and that

1. ***Start LabVIEW***[1] by accessing your lab computers and using the following path: *T:\courses\es50\Labview\lab4-good\lab4*. Take a minute to look at the program and understand the meaning of various panels (boxes) that pop up. For example, in "*Quantized Signal*" box DC voltages brought to Analog Input of the DAQ board will be converted to digital values. Value in "*Number of Bits*" box controls the numbers of bits used in quantizer, and so on.
2. ***Connect DC power supply to the analog input*** of your box and turn it on. Make sure the resolution is set to 8 bits, and that the 'Record' and 'Play Waveform' buttons are off. Start the program by pressing the 'PLAY' button located at the top left corner of the LabVIEW interface and start adjusting the voltage output of the power supply. The program will now show the result of digitizing the analog input voltage with 8-bit resolution.
   a. Change the value of the input voltage from the power supply. Approximately to what analog voltages correspond following the digital values "11111111", "11001100" and "10011001"?
   b. Measure the "step size", in volts, between adjacent changes in the digitized value of the input signal voltage?

   Note: Whenever you wish to look at the signal or update any sampling rates, etc., make sure to stop (the 'STOP' button) LabVIEW, not 'PAUSE' the program. Then to restart the program, press 'PLAY'.
3. Set the input voltage to some desired value and ***reduce the number of bits*** used in the quantizer. Try at least two different number of bits values. Make sure to press the PLAY button for these changes to become active.
   a. What effect does this have on your recorded digital values?
   b. What happens to the step size in the quantizer? Is it more or less precise now?
4. Now that you understand the impact of ADC on DC signals, ***let's take a look at the AC signals.*** Remove the DC power supply from the interface box connections and replace it with the function (pulse) generator. Set the function generator to produce a sine wave with a frequency of 1 kHz and an amplitude of around 1.0 V peak-to-peak. In the program window, set the sampling rate to 5 kHz and the number of samples read to 1,000. Press the 'PLAY' button. Again, make sure the record and play waveform buttons are off. Again to stop sampling, press 'STOP', not 'PAUSE'.
   a. On the graph, zoom in on one of the sine wave periods by right clicking on the graph window, and then selecting *VisibleItems*$-$ > *GraphPalette*. Select the middle button that looks like a magnifying glass, choose the top middle button, and select the portion of the x-axis that you wish to magnify. Sketch it.
   b. How many samples are taken over one period of the sine wave? Does this match the number theoretically predicted from the acquisition parameters?
5. Now let's look at what happens when we ***reduce the sampling rate***. Run the acquisition at sampling rates of 3 kHz, 2 kHz, 1.5 kHz, and 1 kHz.
   a. Sketch one period of the wave at each sampling frequency. Qualitatively, what happens as the sampling rate is decreased?

b. How does the frequency of the sampled signal change as the sampling rate changes? What is the lowest sampling rate at which the sampled signal maintains the original signal's frequency? What is the relationship between this rate and the original signal's frequency?

6. ***Now let's use ADC to digitize more interesting signals – music!*** Insert one end of the audio cable into the audio output of the computer (or your laptop or phone), and connect the other end to the DAQ input (you may need to use alligator clips).

   Set the sampling rate in LabView to 5 kHz and the number of samples to 25,000.

   Make sure LabVIEW is stopped by pressing 'STOP'. Start your music, then hit the 'RECORD' button followed by the 'RUN' button. Wait a bit, to record 5-10 seconds of music, then hit the 'STOP' button. Toggle the Record button 'OFF', toggle the 'Play Waveform' button 'ON', and toggle the 'Display Recorded Signal' button 'ON'. Hit play to verify that your recording worked and hit 'STOP' after you have heard one cycle the recording. If you have difficulties with finding the appropriate buttons, or with anything else, consult your TF (Always a good idea!).

   Important: if you are recording and playing from the same computer you will have to unplug the audio cable to hear the playback!

   Now, record the same audio at sampling rates of 8 kHz and 10 kHz. (To record another audio, stop LabVIEW and make sure 'Play Waveform' is off and 'Record' is on. Then hit play.) Play back the sound samples in each case.

   How does the sampling rate affect the "quality" of the playback sound? Why?

7. The bottom half of the screen displays the digitized waveform of the input. On the left of this display, you can change the resolution in bits and the full-scale voltage range. To record the waveform in this digitalized format, toggle the 'Record as Digital Sound' button and then record the signal using previous procedure. Try this with two different bit values.

8. To ***examine the effects of quantizer resolution on the sound quality***, we will play a little game. Ask your partner to turn around (or leave the bench) while you record a sample of a popular song using 10 kHz sampling rate and 1 bit resolution. Play the sampled song back to your partner and ask her/him to guess the song. If she/he cannot figure it out, then repeat the process using 2-bit, 3-bit, 4-bit, etc… quantizer, till she/he recognizes the song.

   a. What is the minimum number of bits you had to use so that your partner can discern the song and understand the lyrics?

   b. **Optional**: you can try the same game using 8-bit quantizer, and play with the sample rate, instead.

4. Simple Resistor-Based DAC

Now that we know how ADC works, we will build our very own DAC and use it to convert digital outputs from the Arduino into analog signals. To remind you, Arduino can output digital voltages, only, that can assume one of two values: 5 V ("HIGH") and 0 V ("LOW"). Therefore, Arduino cannot output analog signals, including your favorite Miley Cyrus' song! However, DAC come to the rescue (Figure 4a): they can convert digital signal at the output of Arduino into an analog signal that can assume any (more or less) value between 0V and 5 V. In a future homework, you will analyize a 4-bit D/A converter, and synthesize a 6-bit version of it. We will test such 6-bit D/A converter in this lab!

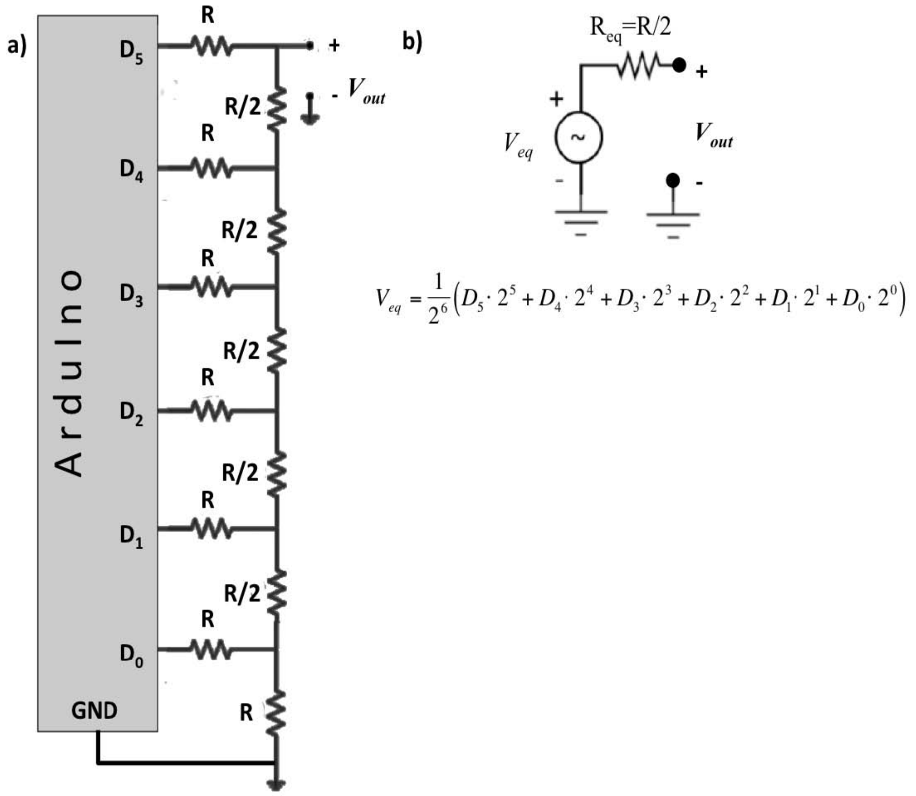
a)
b)
Arduino
D5
D4
D3
D2
D1
D0
GND
R
R/2
+
- V_{out}
R_{eq}=R/2
+
-
V_{eq}
V_{out}
$V_{eq} = \frac{1}{2^6}\left(D_5 \cdot 2^5 + D_4 \cdot 2^4 + D_3 \cdot 2^3 + D_2 \cdot 2^2 + D_1 \cdot 2^1 + D_0 \cdot 2^0\right)$

Figure 4: a) 6-bit D/A converter that consists of a R -R/2 ladder hooked up to Arduino's digital pins 0, 1, 2, 3, 4 and 5. In our case $R = 2\ k\Omega$. b) Equivalent circuit to that shown in panel a).

1. To save you some time, ***nice TFs have already wired-up the DAC circuit*** for you :). However, inspect it and make sure that it corresponds to the schematic shown in Figure 4a. The DAC consists of R-R/2 ladder, and in our case $R = 2\ k\Omega$ (and $R/2 = 1\ k\Omega$, duh…. we used blue 1 kΩ resistors to make it easier for you to follow the circuit). In Figure 4b we show the equivalent circuit: you will derive Veq in an upcoming Homework, so that equation shuold make sense; $R_{eq} = R/2$ can be obtained by grounding all inputs and finding the equivalent resistance as seen from the output.
2. Now let's test our DAC! Connect the DAC to Arduino as illustrated in Figure 4a (make sure that you connect the ground potential to GND!), and hook up the voltmeter to measure V_{out}
3. . Download the *DACtest.ino* Arduino sketch from the course website, and make sure you understand it. Modify the code to send different values of 6-bit binary number $D_5D_4D_3D_2D_1D_0$ to Arduino outputs and record the measured values of V_{out} in each case. In particular, find V_{out} when $D_5D_4D_3D_2D_1D_0$ assumes each of the values 000000, 000001, 000010, 000100, 001000, 010000, 100000, and 111111. How does the measured V_{out} compare to the values predicted by the equation shown in Figure 4b?
4. Next, let's generate some AC signals. Modify the *DACtest.ino* sketch so that it generates a digital sequence that consists of: 000000 for 10 ms, 100000 for 10 ms, 101010 for 10 ms, 111111 for 10 ms, and 001111 for 10 ms. The sequence then repeats itself. Replace the voltmeter with the oscilloscope and observe V_{out} waveform that you've generated. Does the signal correspond to what you've programmed?
5. The last exercise has hopefully convinced you that by sending right digital sequence to the

final project, in fact!), for example. To test this idea, let's generate a bit more complex analog signal - the function $V(t) = \frac{1}{2}[\,1 + cos(2\pi f t)]$ shown in Figure 1. Download the *DACtestCosine.ino* sketch, analyze it, and make sure that you understand each line of the code. If you have questions, talk to your TFs. Once ready, upload the sketch into your Arduino and observe the V_{out} signal on the scope. You should see a nice cosine function! Btw, what determines the period of this signal?

6. (**BONUS: 2 points**) You may have noticed some "glitching" in our output signal -the appearance of various random spikes. These spikes are the high-frequency components of the signal, generated by Arduino, and they can be filtered out. How would you do this? Modify the circuit to get rid of the glitches.
7. Now, let's play with the circuit a bit to see the effects that various changes have on the output signal. For example, unplug D_5 and note down what happens with V_{out} on the scope. Put D_5 back in, and unhook a few more things. Write down your observations. Which change makes the largest impact on the signal, and why?
8. Next, let's play with other parameters of our code. For example, reduce the number of samples N from 100 to 10, upload the sketch, and see what happens. Make sure that you zoom-in on the signal (change the time resolution, if needed) to see all the changes to the waveform. Feel free to play with other values of N, as well as M.
9. (**BONUS: 2 points**) Now that you are expert with D/A converters and Arduino coding, it is time to write your own code to generate a *saw-tooth* signal (Figure 5) at the output. The simplest way to do this is to introduce a variable that is incremented by 1 in a loop ("for" loop can come in handy -check http://arduino.cc/en/Reference for more details), and whose value is output to pins D_5 through D_0. Give it a shot.

   Note: if you don't know how to do this, but are really curious about the code, you can download *DACtestSawTooth.ino*. Of course, you will not get BONUS points in this case…

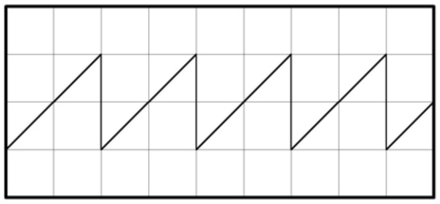

Figure 5: Sawtooth signal