Data Acquisition with the PXIe-4492

By Nicholas D'Anna

Abstract

The National Instruments PXIe-4492 is a Dynamic Signal Acquisition device offering 8 ADC channel simultaneous readout with 24 bit resolution and a maximum supported sampling rate of 204.8 kS / s. LabWindows, a C programming environment and National Instrument hardware API, is used to set 4 ADCs to simultaneously read the outputs of 2 SR830 Lock-In Amplifiers using dc coupling. This report concludes to effectively use the full power of 24 bit resolution, one must consider the temperature varying behavior of ADC pedestals, the errors introduced with anti-alias filters, the errors inherent to floating-point number systems, and the noise produced the environment.

Introduction

Ignoring the more complicated world view quantum theory suggests, we view most physical quantities as continuous over time. In discussing any physical quantity Q, we often select a measurement scheme mapping Q to a real number q. However, all measurements admit error; so, in practice q is always rational. Consider a physical system S admitting an input I and producing an output O. Suppose the output O is a voltage. To analyze O, we must use an analog

to digital converter (or ADC). The task of an ADC is to measure an analog signal (often voltage) and convert it into the closest number belonging to the internal floating point number system of the ADC [1]. Examples of systems using ADCs in our Phys 406 POM classroom include readout of acoustic holography scanning of drum heads and measuring the complex impedance and sound intensity of acoustic, brass, and wind instruments [2].

The error associated with the ADC quantization is not conceptually related to the noise in O. Thus for an ADC with insufficient resolution, truncation error is the dominant error source. To show this, we must discuss the number representation scheme an ADC uses. Modern ADCs (as well as modern computers) use floating-point representation to store number. A floating point number system encodes each number as a series of binary digits (or bits). Some of the bits form the mantissa, m, and the rest form the exponent, E[1]. Floating-point stores a number in the form $\pm m \times 2^E$ where $1 \leq m < 2$. The term floating refers to the number system's ability to place the radix point (decimal mark) at different locations for different numbers in the system [3]. This representation scheme allows a relatively small number of bits to encompass a large range of numbers! For

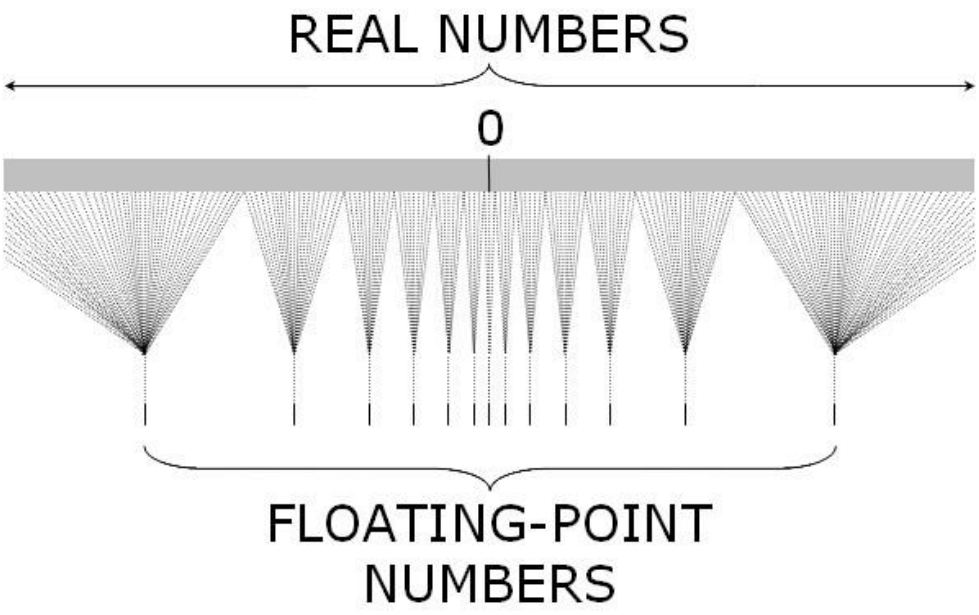

FIG. 1 - A map from a floating-point number system onto the real numbers. The spacing between adjacent float numbers increases monotonically with their magnitude. Image rights belong to [4].

[1] The mantissa and exponent field each use 1 bit to flag their sign.

example, the smallest magnitude (nonzero) numbers an n-bit system can hold naïvely[2] are $\pm 2 \wedge [-(2^{(n-2)/2} - 1)]$ where the n – 2 indicates using 1 bit for the sign of the mantissa and another for the mantissa[2]. We divide by 2 because the first half of numbers in the exponent field map into negative numbers and the second half map into positive numbers. Meanwhile, the largest number that an n-bit system can hold is $+1 \times 2 \wedge [+(2^{(n-2)/2} - 1)]$. Thus the range of an n-bit floating point number system maximally scales up as the 2 to the 2 to the n bits (assuming all bits go to the exponent field). So, upgrading from, for example, 16 bit to 24 bit ADCs as the Phys 406 POM recently has marks a major improvement in the quality of measurements that can be made (see Figure 2).

FIG. 2 - A map from a floating-point number system onto the real numbers. The spacing between adjacent float numbers increases monotonically with their magnitude. Image rights belong to [4].

Note that by allocating more bits to the exponent field, for instance, we sacrifice bits in the mantissa field. The Institute of Electrical and Electronics Engineers (IEEE) defines a standard designation for how many bits should go to the mantissa and exponent. 32 bit numbers, for instance, are defined as single-precision words and use 1 bit for the sign, 8 bits for the

[2] In practice, the first bit of the mantissa may be excluded since it is always 1. This is called hidden bit representation. Some systems also reserve bits for special cases like infinity or NaN.

exponent, and 23 bits for the mantissa. Products utilizing the IEEE standard save their users the effort of thinking about bit representation. However, users of floating-point number systems should not ignore every detail of number system implementation.

Consider carefully the definition of a number in a floating-point system. Note that as the exponent field increases in magnitude, each bit in the mantissa represents a larger number. While the exponent field grants floating-point number systems their flexibility, it also creates a problem: varying precision across the number range. As one may judge by Figure 1, the spacing between neighbors in a floating-point number system increases monotonically with their magnitude. That rounding and truncation errors are a function of number magnitude is the inevitable price to pay for the flexibility offered by float numbers.

The PXIe-4492 is a National Instruments Dynamic Signal Acquisition (DSA) device with 24 bit ADCs and a supported maximum sampling rate of 204.8 kS / s (kilosamples per second) [5]. The PXIe-4492 is advertised as being suitable for "Audio testing, Acoustic measurements, Environmental noise testing, Vibration analysis, Noise/Vibration/Harshness measurements, Machine condition monitoring, Rotating machinery evaluation" (National Instruments, 1-1). We characterize how the PXIe-4492 quantizes analog signals with several properties including input channel configuration and input coupling. Knowledge of these properties is crucial if one is to properly take data. The PXIe-4492 only supports the pseudodifferential input channel configuration, which is best reserved for measuring floating signals. A floating signal (or nonreferenced signal) is one without any ground reference. The pseudodifferential configuration provides a ground reference between the floating signal and the PXIe-4492 by connecting a 50 Ω resistor from the negative input to the ground. This connection prevents the floating source from drifting beyond the measuring range. Unfortunately, a pseudodifferential configuration creates

two grounds for grounded signals and that may encourage ground loops, resulting in noise. The 50 Ω resistor helps abate ground loop noise, but its ability to correct the problem depends largely on the experimental setup.

Input coupling in the PXIe-4492 is either set to read just the ac component of a signal or the signal and its dc bias. The PXIe-4492 switches between ac and dc coupling by adding or removing a resistance in parallel with a capacitance. Consult Figure 3; if the resistor R has an infinite resistance then the signal is forced through the capacitor C. When the dc bias component of the signal reaches C, it starts to charge C's plates until their voltage magnitude matches the magnitude of the voltage from the input and the dc current flow stops. Since the ac component changes, the charge on the plates of C change correspondingly. This changes amounts to transferring the signal across C, uninterrupted. To switch to dc coupling, R changes to a zero resistance shorting out the capacitor and allowing the full signal to pass. When the PXIe-4492 switches input coupling, it takes about 5.5 seconds to settle into 24 bit accuracy.

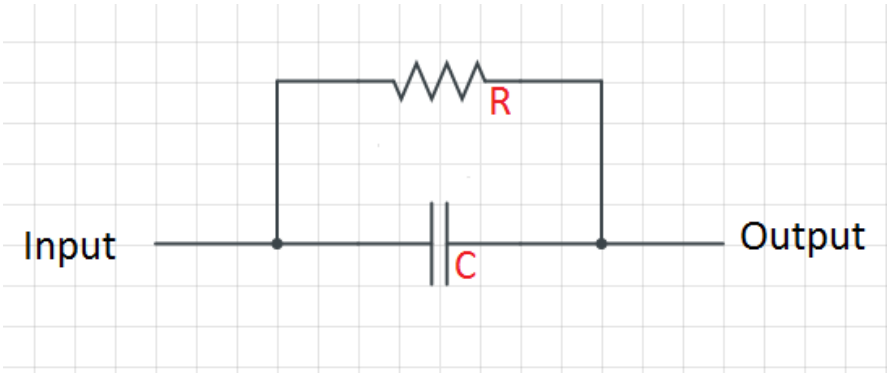

FIG. 3 - A heuristic schematic of the circuit responsible for switching between input couplings. R is a variable resistor equal to 0 Ω for dc coupling and ∞ Ω for ac coupling.

The PXIe-4492 may introduce spurious from aliasing. This results from limitations in the bandwidth of the signals it can measure. Suppose one sets the ADC to a sampling rate of 2**f**; the associated Nyquist Frequency is f and the ADC cannot measure signals with frequencies greater than **f** [5]. Suppose a signal contains at least one frequency component falling outside of the

Nyquist bandwidth. The digitizer in the ADC may construct artifacts from these components (aliasing). Aliasing is extremely problematic because it's difficult to determine whether or not it occurred. The PXIe-4492 incorporates the use of both digital and analog low pass filters to screen out frequencies above the Nyquist bandwidth. For the loss pass filter system to work, one must adhere to some idiosyncrasies of the hardware. For instance, if a sampling rate **f** between 102.4 kS/s and 204.8 kS/s is used, readout is compromised for signal frequencies in frequency bands of width **f** centered at 32**f**, 64**f**, …, 32n**f**. That is, the frequency bands given by [32n**f** – **f**/2, 32n**f** + **f**/2] for all natural numbers n, admit aliases. Similar rules exist for lower frequency ranges.[3] One way to mitigate aliasing problems is to select a sampling rate high enough that the first aliasing bandwidth lies entirely above the low pass filters cutoff point.

I close the Introduction section with a brief discussion of the SR830 lock-in amplifier as the PXIe-4492 will often collaborate with it. The SR830 implements 2 interfaces for remote programming: the RS323 or the IEEE-488.1 standard GPIB interface [8]. This project used the GPIB interface. There are 3 steps for issuing a command through the GPIB interface on the SR830: set a C string S to have only null character ('\0') entries; write the command name and quantity applying to the command to S; and send S to the GPIB interface. The last step is taken care of using National Instruments's API. The first step should always be done otherwise the GPIB interface may interpret garbage data grabbed by the compiler when the C string was declared. The Methods section explains this process in more detail.

[3] For sampling frequency **f** between 51.2 kS/s and 102.4 kS/s, users should avoid measuring at multiples of 64**f**. Sampling frequencies lower than 51.2 kS/s will have problems reading multiples of 128**f**.

Methods

National Instruments programming environments (LabVIEW and LabWindows/CVI) define, as part of their APIs, commands for executing measurements with the PXIe-4492. For this project, we elected to use LabWindows/CVI which uses the C programming language. In LabWindows, communication with the PXIe-4492 generally involves 6 steps: creating a task, configuring a channel, starting a task, performing read(s), stopping a task, and clearing a task [5]. A task is a programming object defined by National Instruments that allows users to specify and take measurements with National Instruments hardware. We created a task using the function DAQmxCreateAIVoltageChan. This function creates a virtual connection to the PXIe-4492 and initializes its properties to their default values (i.e. input coupling is set to ac). Next, using the generic function DAQmxSetChanAttribute, channel properties such as sampling rate, input coupling, and number of channels to read with in parallel must be set. For our purposes, we set up 4 ADCs to read simultaneously at a rate of 200 kS/s with dc coupling. After this, we call DAQmxStartTask to commit all channel attribute changes to the PXIe-4492 and prepare the system for performing measurements. It is important to follow the steps mentioned so far as infrequently as possible, otherwise the application must wait unnecessarily for the dc coupling to settle to 24 bit accuracy with each measurement. Within our main measurement loop, we call DAQmxReadAnalogF64 which reads out each selected channel simultaneously and stores the data into a float64 array. When we finish taking measurements for the program run, we call DAQmxStopTask to tell the PXIe-4492 to exit sampling mode. At last, we clear the task from memory.

Remote control of the SR830 lock-in amplifiers is more straightforward. The SR830 executes commands by parsing what I will refer to as command strings. A command string is a C string consisting of a 4 letter mnemonic and a space delimited parameter list. In the Results section, we take a look at Czerepak's project [7]. That study only requires 2 commands: "FREQ *f*" and "SLVL *a*" where *f* and *a* are numbers [8]. FREQ and SLVL set the lock-in amplifier's internal oscillator, respectively, to a frequency of *f* and an amplitude of *a*.

What follows is a pseudo code description of the algorithm developed for use in Czerepak's project [7]. It demonstrates how to follow the trappings of successful measurement with the PXIe-4492 reading out from the SR830:

```
PROGRAM: FREQUENCY_SCAN(MIN_FREQ, MAX_FREQ, STEP_SIZE, AMP)

1    Declare samples <- Input
2    Declare ADC0RawData [samples],…, ADC3RawData [samples]
3    Declare size <- (MAX_FREQ – MIN_FREQ) / STEP_SIZE + 1
4    Declare ADC0Means, …, ADC3Means [size]
5    Declare ADC0Sigmas, …, ADC3Sigmas [size]
6    Declare task <- Create_Task() % Assign memory for task
7    Create_Voltage_Channel(task)  % Virtual connection
8    Set_Samples_Per_Frequency(task, samples)
9    Set_Input_Coupling(task, dc)
10   Set_Channels_To_Read(task, ADC 0 – ADC 3)
11   Declare Vlockin <- Input
12   Declare cmdString[50]
13   cmdString <- 50 * '\0'   % fill each element with null
14   cmdString <- "SLVL <Vlockin>" % where <> evaluates var
```

```
15   Write cmdString to GPIB interface
16   Declare i <- 0
17   for f from MIN_FREQ to MAX_FREQ by STEP_SIZE
18       Declare attempts <- 0
19       do
20          attempts <- attempts + 1
21          cmdString <- 50 * '\0'
22          cmdString <- "FREQ <f>"
23          Write cmdString to GPIB interface
24          Wait()
25          [ADC0RawData,…, ADC3RawData] <- Read_ADCs(task)
26          ADC0Means[i], …, ADC3Means [i] <-
                Get_Means(ADC0RawData,…, ADC3RawData)
27          ADC0Sigmas[i], …, ADC3Sigmas [i] <-
                Get_Standard_Deviations(ADC0RawData,…)
24       while (ADC0Sigmas[i] > SIGCUT0 OR … > SIGCUT4 AND
                 attempts < 100)
25        if (attempts == 100)
26             Save_Data_And_Quit()
27   Save_Data_And_Quit()
```

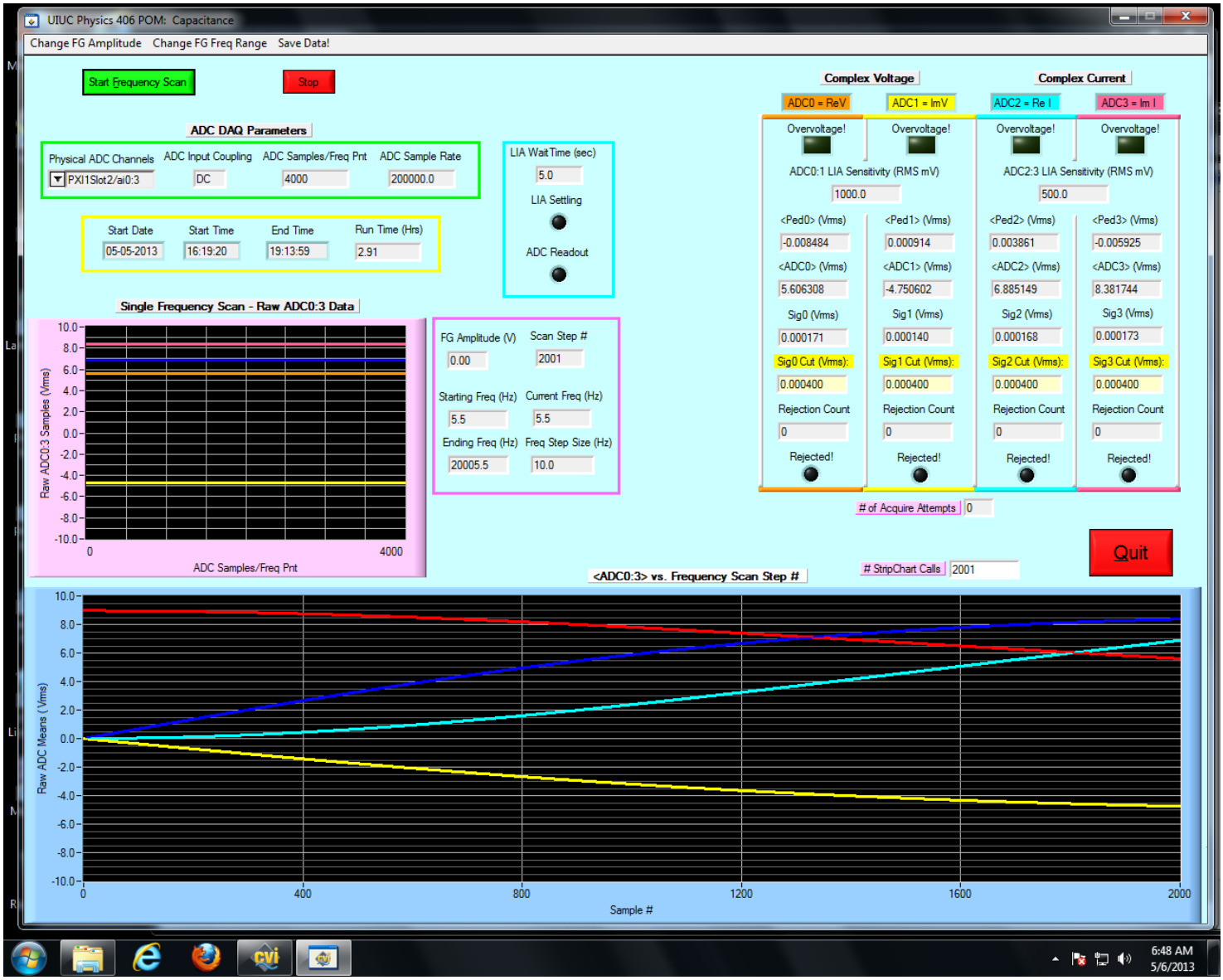

FIG. 3 - The finished application display panel. The raw data graph is the pink box and the mean data is displayed in the cyan box. The other indicators display various values of interest such as the measurement number (i in the pseudo code), pedestals, and whether or not a measurement made the sigma cut.

Implementing this pseudo code into a LabWindows application is the main activity behind my project. I want to comment on some lines of the above pseudo code for clarity. Lines 6-10 comprise the first 3 steps for performing measurements with the PXIe-4492. The loop on line 17 sweeps through a frequency range. At each frequency, the application prompts the PXIe-4492 to simultaneously read out the 4 channels specified in line 10. Line 24 is very important to the precision of data: it allows the lock-in amplifier to settle after switching frequencies. Arguably the most important check performed by the code is line 24. Line 24

watches to see if some local variation occurs because of a disturbance in the system. If a standard deviation larger than some preset occurs, the application retakes data at that frequency point until it falls back into the realm of normalcy. The pseudo code is a bit of an idealization, but illustrates the main challenges of the code. Figure 3 shows the finished application after a run. Note the real code uses pedestals to correct for noise on the ADCs. We have documented[4] the pedestals as functionally depending on the room temperature and the line voltage to the device. If either of these quantities change throughout a pedestal run or after a pedestal run yet before your actual data taking ends, error proportional to these fluctuations results. Another thing not shown by the pseudo code performed in the actual application is a software level check that we shut off the application if voltages approach levels that may damage the ADCs. Although the hardware claims to be capable of this automatically, we still decided on placing this safety in the software. The real application also plots data so that users have an idea of what measurements occur the moment they look at their computer.

[4] This is easy to see. Set the equipment up for a pedestal run (terminate each ADC with resistors) and start the run just before sunset. By analyzing the mean graph, variations will be evident as the room cools.

Results

This section revolves around Anna Czerepak's project and the complications arising thereof [7]. Czerepak's project involves feeding a circuit output into lock-in amplifiers. The PXIe-4492 reads the lock-in amplifiers' output. After debugging the PXIe-4492 application fully, our first result contained an unexpected amount of noise. It turned out the lines going into the PXIe-4492 contained a negligible amount of insulation and the display screen of the lock-in amplifiers disrupted the measurement signal. The setup is shown in Figure 4.

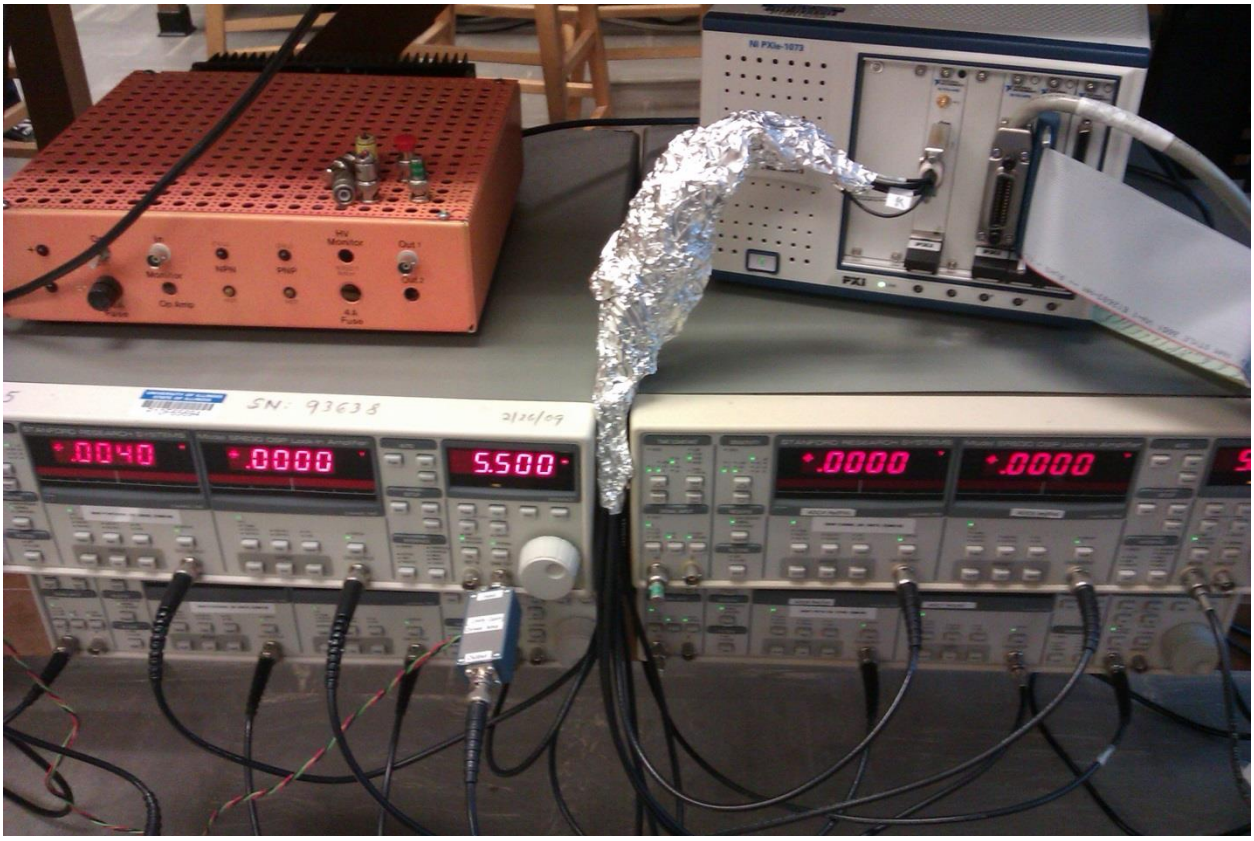

FIG. 4 - Although there are 4 lock-ins in this picture (2 on top, 2 below) only 2 of them are attached to the PXIE-4492. Although it may look sloppy, the tin foil wrapping is crucial to the accuracy of our measurements as the ADC lines had grossly insufficient insulation from environmental noise.

As Figure 4 shows, we eventually wrapped tin foil around the ADC lines. After doing so the noise dropped dramatically. After this remedy, we gathered some pedestals for the system. The application output was:

```
Impedance_Pedestals_05-08-2013.txt

 RAW ADC means and sigmas:
 =========================
 <ADC0> =    -0.008611 +_  0.000026 (RMS Volts)
 <ADC1> =     0.000763 +_  0.000010 (RMS Volts)
 <ADC2> =     0.003758 +_  0.000015 (RMS Volts)
 <ADC3> =    -0.006110 +_  0.000040 (RMS Volts)

 RAW ADC Sigma means and sigmas:
 ===============================
 <SIG0> =     0.000151 +_  0.000002 (RMS Volts)
 <SIG1> =     0.000150 +_  0.000002 (RMS Volts)
 <SIG2> =     0.000147 +_  0.000002 (RMS Volts)
 <SIG3> =     0.000146 +_  0.000002 (RMS Volts)

 RAW ADC COUNT means and sigmas:
 ===============================
 <ADC0> = -7223.162047 +_ 21.903917 (RMS Volts)
 <ADC1> =   639.865076 +_  8.782895 (RMS Volts)
 <ADC2> =  3152.630051 +_ 12.892912 (RMS Volts)
 <ADC3> = -5125.062609 +_ 33.425919 (RMS Volts)

 RAW ADC COUNT Sigma means and sigmas:
 =====================================
 <SIG0> =   126.435732 +_  1.663042 (RMS Volts)
 <SIG1> =   125.631667 +_  1.496389 (RMS Volts)
 <SIG2> =   123.595092 +_  1.569922 (RMS Volts)
 <SIG3> =   122.523983 +_  1.435507 (RMS Volts)

% n.b. LSB of NI PXIe 4492 ADC for +_10V dynamic range:
%      20V/2^24 = 20V/16777216 = 0.0000011921 V = 1.1921 uVolts.
```

Conclusions and Future Work

The PXIe-4492 application seems to be in order. We still need to make some minor adjustments to set up 8 channel readout, but the actual algorithm may be left alone. In the results of Czerepak's project, there are some unexplained peaks in the data. This indicates that something is going wrong with the readout. Over the summer I plan on taking a closer look.

References

[1] “Digitation of Analog Quantities.” May 1, 2013. < http://iamechatronics.com/notes/general-engineering/279-digitization-of-analog-quantities>

[2] Errede, Professor Steven (2013). Physics 193POM/Physics 406POM Acoustical Physics of Music/Physics of Musical Instruments Lab Setups/Lab Equipment. http://courses.physics.illinois.edu/phys406/sp2013/Lab_Handouts/P406POM_Experiment_Setups.pdf

[3] Anne Greenbaum & Timothy Chartier (2012). Numerical Methods: Design, Analysis, and Computer Implementation of Algorithms. Chapter 5: Floating-point Arithmetic.

[4] Izquierdo, Luis R. and Polhill, J. Gary (2006). 'Is Your Model Susceptible to Floating-Point Errors?'. *Journal of Artificial Societies and Social Simulation* 9(4)4 <http://jasss.soc.surrey.ac.uk/9/4/4.html>.

[5] National Instruments (November, 2010). NI Dynamic Signal Acquisition User Manual.

[6] National Instruments. http://www.ni.com/pdf/manuals/372125f.pdf. Accessed 05/09/2013

[7] Czerepak, Anna (2013). Measurements of Capacitor Noise.

[8] Stanford Research Systems (2005). Model SR830 DSP Lock-In Amplifier Manual.