This is documentation for the DAQ (Data AcQuisition) code written by me (Cass). My code is one instance of a series of attempts over the past few years to get a functional system capable of reading voltages and outputting voltages.

It is, in its current state, woefully incomplete; as far as I am aware, it is also the only one that works.

I’m writing this as a basic explainer (similar to what I’d give in-person) for what the DAQ is and how it works. Code documentation will follow later.

Every instance of the DAQ has had four major components:

* An analog to digital converter (ADC) to read voltages
* A digital to analog converter (DAC) to output voltages
* A controller to handle communication between the ADC, DAC, and a computer
* A computer to process the data output

That’s it! It’s the sort of system that I find fundamentally pretty simple until you get into the details.

The original DAQ was an attempt to recreate a system based on this: <https://opendacs.com/> The lab spent years trying to get this system to work, and I’m frankly not sure what’s gone wrong each time. As such, when I started to work on the DAQ I basically threw everything out and started from scratch.

Here’s all the hardware configurations we’ve tried:

| System | ADC | DAC | Controller |
| --- | --- | --- | --- |
| Original system | [AD7734](https://www.analog.com/en/products/ad7734.html) | [AD5764](https://www.analog.com/en/products/ad5764.html#product-overview) | Arduino Due |
| Cass (1st try) | AD7732 | AD5764 | Ti MSP432 |
| Cass (2nd try) | [AD7606](https://www.analog.com/en/products/ad7606.html) | AD5764 | Raspberry Pi |
| Future plans | AD7606 | AD5764 | Arduino Due/Portenta |

The lab has *so* much documentation for the original all over the place. There are a few lab notebooks sitting at the DAQ bench with notes. There’s some notes on the google drive as well:

<https://drive.google.com/drive/u/0/folders/1pOTZbaR7F627t7Xn4o5nZmigleL3lWHf>

Many code examples from other groups are also available on the google drive.

I, on the other hand, am notoriously awful with documentation, so there isn’t much written down from me aside from this document. I’m working on getting better!

You can see from the table that the DAC has always been the same. The AD5764 is a 16-bit DAC with four output channels. It’s a simple device and has basically done everything we’ve needed it to.

The ADC has changed twice. The AD7734 and AD7732 are effectively the same chip with a different number of readout channels. They’re both high precision ADCs (24-bit), however they come with the disadvantage of low bandwidth (<2kHz). The AD7606 on the other hand is considerably faster (1MHz), but less precise (18-bit). There’s a case to be made that using both might be useful so we can get both the high precision and the high bandwidth (not at the same time, obviously), but that’s not something we’re concerned with right now.

The controller changes aren’t that significant, but still are worth mentioning: we currently have an arduino and a raspberry pi in the lab. The MSP432 is a device that I personally owned as part of a class I was taking. It’s effectively a more professional arduino: you can code for it in the arduino IDE and everything- it just has more features. It is also broken! Some static discharge broke some of the output ports. The use of a raspberry pi as the controller was driven by my desire to have the controller and the computer be the same device to avoid any bottlenecks in serial communication. (I also dislike coding in C++.)

The software side of the DAQ has to complete an assortment of tasks. It must:

* Acquire data from the ADC and DAC over SPI
* Communicate the data to a desktop computer (if applicable)
* Process the data (FFT, etc)
* Display the data

There are a myriad number of ways to write code that completes these steps. While the choice of hardware limits some of the options, there are many ways to write a GUI, communicate over serial, etc. The most feasible options are the ones that have pre-existing codebases, libraries- anything that reduces the amount that we need to code ourselves.

To that end, the code we are developing falls into two (broad) categories: firmware and software. Firmware is specifically code that will be run on a microcontroller, whether it is an arduino or a different device akin to the MSP432. Software is code that runs on a typical desktop computer (and will be effectively synonymous with “code”).

Regardless of the microcontroller chosen, firmware will nearly always involve writing C++ code for a device that communicates with the ADC and DAC. I’ve looked into two methods to avoid writing dedicated firmware, but both methods have their flaws:

**rPi**: The raspberry pi very conveniently combines the application options and general flexibility of a desktop computer with the GPIO options necessary to communicate with the ADC/DAC.

The major advantage of having all necessary tasks handled by one device is that it dramatically decreases the complexity of the project. All the rPi code can be written in python. There’s no need to worry about communicating the data because it’s all on one device. The hardware itself is powerful enough that it doesn’t require external processing by a more powerful computer.

The disadvantage is that the process of gathering the data is slow. SPI communications have to go through the linux kernel which delays the transaction (roughly 0.1ms). The rPi also lacks a real-time clock, which means that it is unable to get points at reliable intervals. Both of these disadvantages make it considerably more difficult to take a reliable FFT on an rPi.

**MATLAB**: MATLAB has a unique method of programming an arduino: through a sketch that effectively acts as a programming server, it can execute programming commands on an arduino directly through a desktop computer. This direct control of the arduino gives it similar advantages to those of the rPi: all the necessary code can be written in MATLAB and the necessary libraries for arduino communication are provided.

The disadvantage is that, again, it’s slow. The time it takes to send a command to start a conversion is an unacceptably long 6ms. This is due to a slowdown in the USB protocol (there’s always something!) which means that there isn’t a workaround for it (such as increasing the baud rate).

It’s worth noting that *the only reason these flaws matter is because of the FFT*. Eliminating the FFT from the DAQ code would make either of these methods viable as we would no longer require fast speeds, however it would also require code to handle an external spectrum analyzer. My assessment of the situation is that this will likely be more of a hassle than writing our own code as the addition of an external device will make the process of gathering data more time-consuming. Should we later decide that we do wish to cut the FFT from our code, I would highly recommend going back to the MATLAB app.

We have a lot of semi-functional code that we can draw from.

* Collaborator code (Andrea, Weizmann)
* Joe code
* Pathiq code
* My (old) code
* Current MATLAB code

Every reference we have (with the exception of my old rPi code) loads a sketch onto an arduino to obtain data from the ADC/DAC. They all differ in how that data is handled.

The collaborators have developed a few different methods of handling the data. They have code for MATLAB, code for LabView, and code in python.

Do not use LabView.

All of these options functionally do the same thing: the desktop code sends a command to the arduino, the arduino receives the command and performs data gathering depending on the command, the arduino sends back the data and the desktop code receives it. The difference is what receives the information back on the desktop.

Joe’s code uses a similar system in python.

I genuinely could not tell you what Pathiq’s code does.

Other people’s code can be used as a reference for how we should write our code. We would need to modify the portions that pertain specifically to the AD7734 and replace them with the AD7606. My opinion is that it would be most useful to make an arduino sketch similar to what others have done and communicate with a MATLAB app on the desktop.