480W Data Acquisition

Quinn Pratt

ver. 1.7

Abstract

This document serves to describe the data acquisition methods for experiments performed in the advanced laboratory course (PHYS 480W). Data acquisition routines are an *essential feature* of modern physics experiments. We explore acquisition through industry standard programs such as MATLAB and National Instruments' LabView.

Contents

1 Introduction

For the purposes of this course, we will focus more on the utility end of data acquisition, as opposed to delving into the nitty-gritty programming concepts surrounding Data Acquisition devices (DAQs). Regardless of the experiment, data acquisition can be pictured roughly as shown in fig 1.

Note: although the data acquisition hardware we will be using has the capability to output analog signals (hardware → actuator), we will not be using this functionality.

We can adapt the diagram above to reflect our system:

- **Sensor:** One example of a data acquisition system sensor would be a thermocouple, a pressure gauge, an ammeter, etc. In our case, the 'sensor' will usually be a voltage reading across a resistor.
- **Signal Conditioning:** this refers to the ability to include filters, voltage dividers, amplifiers, or other post-sensor analog or digital instruments.
- **Acquisition Hardware:** In our case, this will exclusively be National Instruments DAQ hardware. examples used in this course will be the NI myDAQ, or the NI USB-DAQ 6343. Acquisition hardware differs from a sensor in that these are special types of hardware which have the ability to communicate with a computer (through a driver).
- **Software:** We will cover two software methods for acquisition, MatLab and LabView. Furthermore, both of these programs make use of the DAQmx driver, however, MatLab additionally requires the Data Acquisition Toolkit.

Although prior experience with MatLab hardware-programming is not required for using the data acquisition routines outlined in this document it is important to note that the scripts and user interfaces are not the whole story, even after acquisition, additionally post processing and analysis will be required, and to that end, basic MatLab competency will be crucial.

2 Hardware

In this section we will discuss the National Instruments hardware which makes computer-automated data acquisition possible. Although data can

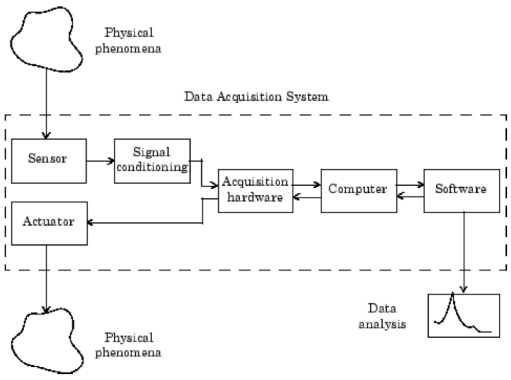


Figure 1: DAQ hardware is a software controlled apparatus that receives input data from a transducer or sensor which is causally connected to the physical phenomena one wants to measure somehow. DAQ hardware permits the extraction of a data stream produced by the sensor, issuing in a file which, through post processing, becomes a graph in ones research paper or report.

be collected through a variety of instruments such as digital storage oscilloscopes (DSO's), the advantage of the NI hardware is that it is designed to be controlled by the computer.

2.1 NI myDAQ

This device is a small, USB-powered data acquisition board. These boards we designed by NI to be easy-to-use academic DAQ hardware, meaning that they are meant to be used by students to probe circuits, attach accelerometers, thermocouples, digital switches and controls, etc. They can only handle input voltages of $\pm$10V, and they have a maximum sample rate of 200kS/s.

Figure 2 is a shot of the NI myDAQ. Note the plug-in terminals along the right side

We have fitted each of these boards with a BNC-add on (figure 3). This converts the plug-in ports on the side of the DAQ to industry-standard coaxial BNC ports. There are 4 BNC ports on the myDAQ, each of them has been hard-wired for single-ended measurements with a ground-reference directly above the ports. the two ports on the left are analog-out (AO), while the ports on the right are the analog-in (AI).

Figure 2: NI myDAQ device. We will use 2 such devices this semester, one labeled Eric and the other Quinn

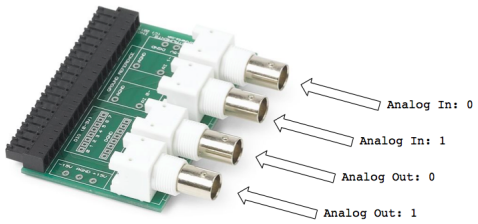


Figure 3: Use of myDAQ is made considerably easier with a BNC connector extension board which snaps into the DAQ.

Figure 4: The NI USB-6343 has 4 analog outputs and 23 analog inputs, with a maximum digitization rate of 500ks/s.

2.2 NI USB-6343

While the myDAQ is a very versatile instrument, some experiments require a more sophisticated data acquisition board. This is the NI USB-6343, shown in figure 4, which is used to send and receive analog and digital signals. It has a maximum sample rate of 500kS/s and is much more power than the myDAQ. It has 23 Analog-In BNC inputs as well as many screw-terminals and 4 Analog-Out BNC terminals.

This device also connects to the computer through USB and can be programmatically controlled through either MatLab or LabView.

3 MatLab

This section demonstrates the use of the NI Hardware in conjunction with MatLab and the MatLab Data Acquisition Toolbox (DAQ Toolbox). We will also cover the Graphical User Interface (GUI) which was developed with the intention of simplifying data acquisition in MatLab.

3.1 MATLAB .m file to control MyDAQ

This subsection describes how to use the graphical user interface which was developed to make Data Acquisition even more user-friendly. The GUI places

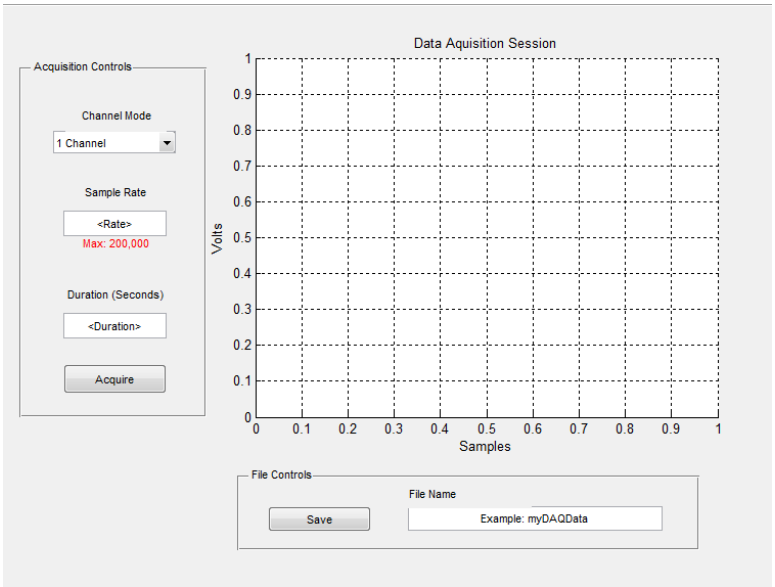


Figure 5: MATLAB data acquisition toolbox generated generic GUI that controls MyDAQ.

a veil over the actual code which is being executed as the user fills out the values and pushes buttons in the separate window. While the details of the code are important, we limit ourselves here to the use of the GUI itself. With MATLAB running, ones opens and runs `myDAQ_480W.m`.

Note: This GUI is only compatible with the NI myDAQ hardware.

When the GUI is executed, a small window will appear (figure 5); this window shows several editable-text boxes as well as a large, empty plot.

To use the GUI, the user should follow the following steps:

1. Select whether the experiment required two channels of data, or one channel.

2. Type in the desired Rate for acquisition an example of a moderate rate would be 10,000 samples/second. The NI myDAQ is limited to 200,000 samples/second.

   Note: The user should type these parameters in the form of numbers ONLY, i.e. do not type 10,000; instead type 10000.

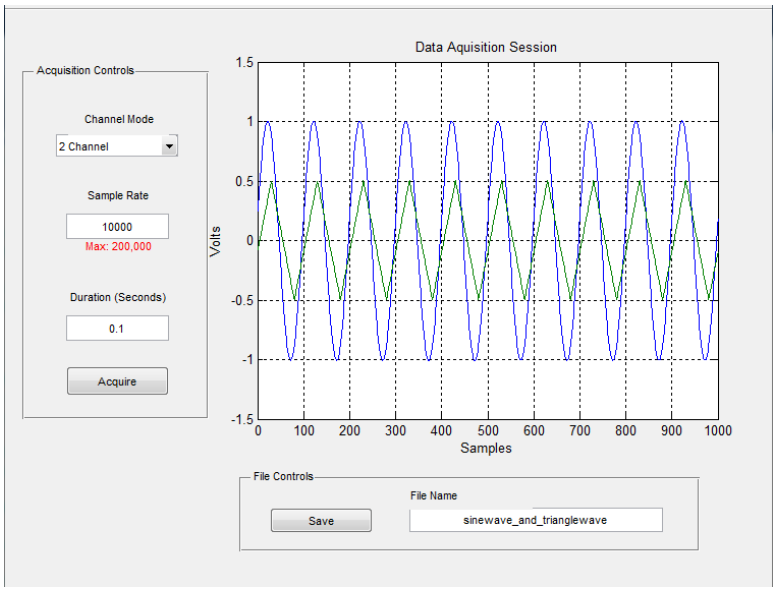


Figure 6: Here, a two channel GUI has been selected, with a digitization rate of $10^4 \#/sec$, and a duration of a tenth of a second.

3. Type in the desired acquisition Duration. Another important quantity is the Rate $\times$ Duration, which tells the user how many samples will be gathered. For example, if the user inputs 10,000 as the rate, and 0.1 as the duration, the acquisition session will collect 1000 samples.

4. Once the properties have been configured, the user can click the 'Acquire' push-button. Each time the button is pushed, the session is executed with the user-inputted properties, and the data is plotted. The user is free to adjust the properties and re-acquire. *Note:* the user **cannot** change between 1-channel and 2-channel freely, this will generate an error.

5. When ready, the user can save the data by typing an appropriate file-name in the editable-textbook at the bottom of the window, and hit the save pushbutton. The save button will produce a .csv file in the current MatLab directory. It is not necessary to include the file extension in the file name, the .csv extension is appended afterward.

For Two-channel sessions it might look like what is shown in figure 6.

Once the user has acquired the data he/she wants, and they have saved the data to a .csv file, the functionality of this GUI has reached its end. That is to say, this GUI environment was not designed for additional post processing. The first step of any post-processing routine should be to use the MatLab code: `csvread('YourFileNameHere.csv')` to input their data as a MatLab array.

The .csv files generated will be of a specific form. The Data Acquisition will place the signals in a samples-by-channel array. where samples are rows, and each channel adds another column. For example, if one were to save the sine wave from the one-channel example, they would see a cdv file with one column (representing the one channel) and 1000 rows (representing the 1000 samples collected). In the two-channel example, the resulting .csv file would be 1000-by-2 with the second column containing the triangle wave data.

4 LabView

Although all of the newest data acquisition protocols rely on MatLab, we make use of the National Instruments' graphical programming environment "LabView" for some of the labs. The analogous thing in LabView to the Matlab GUI is the 'instrument.vi' which mimics the data acquisition instrument under control of LabView software. A screen shot of `Grab Trace 480w.vi` is shown in figure 7, a simple .vi file which was written to mimic the utility of a generic 4 channel digital storage oscilloscope.

LabView is often credited as the industry standard for automation and data acquisition. There is a reason why we use National Instruments hardware (myDAQ and the USB-6343). LabView is the software to go along with the hardware. The traditional data bus for software instrument control has been the IEEE 488 standard or the general purpose interface bus, or gpip. Most pc installations have a gpib card installed in the pc and instruments have been typically connected to the pc with gpip cables. There are now 'gpib to usb' connectors to obviate the need for the gpib card to be physically installed inside the pc box or lap top. This data circuit is shown in figure reffig:labviewcontrol. The urgent point here is that one could control a DSO or a function generator or a lockin amplifier, power supply, pulse generator, or DAQ board in this fashion.

Like the Matlab GUI, the virtual instrument, or .vi file is really the result

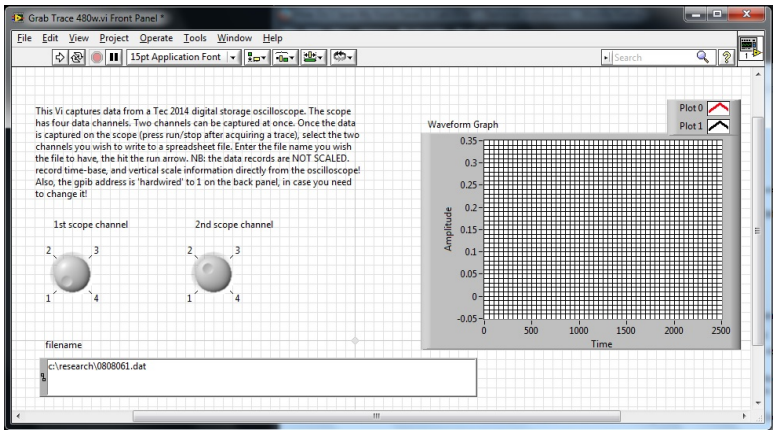


Figure 7: A generic 'virtual instrument' front panel as it would appear in a window generated when the .vi file is launched, which simulates a 4 channel DSO. In the spirit of full disclosure, the actual gpib address (7 instruments are readily addressable) is hardwired with a string constant in the back panel of the .vi file.

Figure 8: Instruments such as the DSO can acquire data using the signal processing present in the instrument itself, and be controlled by LabView software via the pc through the gpib bus, using the hardware shown above. Gpib cabling is generally required.

of graphical coding which can be created and edited in the so-called back panel of the GUI. LabView uses a graphical language, meaning functions and variables are represented not by words but by little images which can be manipulated on the so-called 'front panel'. It's the front panel that is coded to look like the physical device one is taking data with. There is a whole lot more too it, and that discussion is deferred to the appendix.

Let's consider how to use the virtual instrument to take data, the use of which is predicated on the prior use of a 4 channel digital storage oscilloscope (TEKTRONICS TDS2014, in our case) to acquire the data. One will want to become familiar with the use of the scope first and configure inputs, scale sensitivities, triggering, and the time base so as to acquire the signal in the first place.

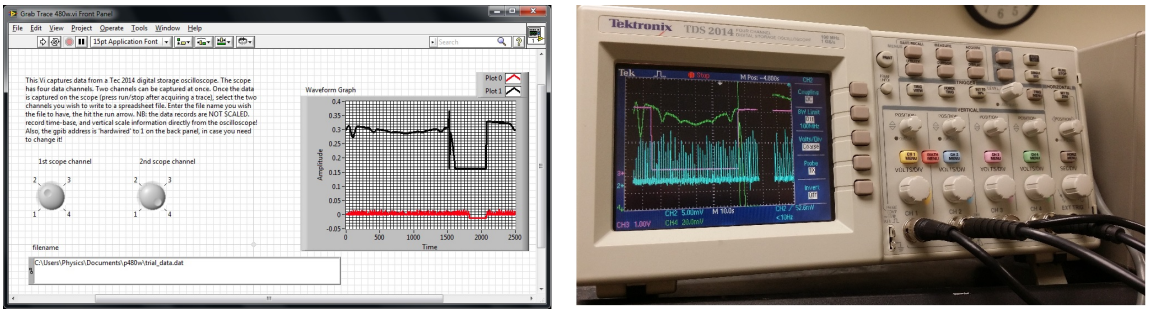
Then,

1. Select which two channels of data from the scope one wants to acquire using the knob controls on the front panel of vi.

2. The details of the digitization rate are determined by the scope settings. In particular, the time base of the scope determines the digitization rate: 2500 points are collected per 10 units of time. So, for instance, if one chooses a time base of 10 sec/box, one has then chosen a digitization at rate of 25 acquisitions/sec.

3. Then one fills in the complete path name of the file to write the file into the appropriate directory.

4. To execute the data capture from the controlled instrument (yes LabView can control lots of different instruments), simply click the arrow button in the toolbar of the vi. The grid pattern on the front panel goes blank and arrow button turns a funny shape, and you wait, and get bored, and then it comes back (the front panel grid pattern) with a graph. Now the odd thing is that it's not always clear which data record forces the auto scale, and you will note that the graph isn't scaled in X or Y as the oscilloscope is, so NB: please record **PLEASE RECORD** the scope parameters for each data file. Or, mess around with G-Programming to hardwire in the scale factors on the displayed graph. You will do something like this in post-processing, probably in MATLAB



(a) (b)

Figure 9: On the left (a) note the image of the front panel of the vi, with the data displayed on the waveform graph. On the right (b) is a picture of the actual device that took the data. The vi is simply a caricature of the actual instrument with only those features the experimenter cares about coded in.

Once done successfully, the face of the vi plots the data. Compare this with the device that actually took the data in figure 9

Once the user has acquired the data he/she wants, and they have saved the data to a .csv file, the functionality of this GUI has reached its end. That is to say, this GUI environment was not designed for additional post processing. The first step of any post-processing routine should be to use the MatLab code: `csvread('YourFileNameHere.csv')` to input their data as a MatLab array.

The .csv files generated will be of a specific form. The Data Acquisition will place the signals in a samples-by-channel array. where samples are rows, and each channel adds another column. For example, if one were to save the sine wave from the one-channel example, they would see a cdv file with one column (representing the one channel) and 1000 rows (representing the 1000 samples collected). In the two-channel example, the resulting .csv file would be 1000-by-2 with the second column containing the triangle wave data.

5 Appendix I: MATLAB Programming Acquisition Notes

Although the GUI is very useful, it is also important to understand how it works. The GUI places a veil over the actual code which is being executed as the user fills out the values and pushes buttons in the separate window.

To understand the background code which communicates with the myDAQ and subsequently gathers data, we must first discuss the relevant functions and properties.

This compatibility between the myDAQ would not be possible without the Data Acquisition Toolbox. This toolbox allows the user to avoid using the more advanced DAQmx C-scripts by bundling the commands into a set compact MatLab functions.

Examples of said functions are:

- `createSession(...)`
- `addAnalogInputChannel(...)`
- `startForeground(...)`

In addition to simply collecting data we must also configure certain properties of our acquisition, examples of configureable properties are:

- `Rate`
- `DurationInSeconds`
- `Name`
- `IsContinuous`

These functions and configureable properties are the core of MatLab data acquisition. However, we have not yet addressed how to call these functions, or how to configure the properties of a session.

MatLab is an object-oriented programming language, simply put, this means that the user must create virtual 'objects' in the computer's memory, then the user can modify said 'objects' using functions. In the realm of data acquisition this means that we must create a virtual object to represent the

action of collecting analog data through a driver... this virtual object is called a *session*.

Sessions in MatLab are objects, just like a matrix, a variable, a figure, a function handle, etc. Any experience the user may have plotting data in MatLab will find this very familiar. Just as with a plot the user can configure whether there's a legend, what the axis limits are, whether the grid is visible or not, etc. The user can configure a session to fit their acquisition needs.

The following is sample DAQ code:

```
s = daq.createSession('ni');
ai0 = addAnalogInputChannel(s,'myDAQ1','ai0','Voltage');
ai1 = addAnalogInputChannel(s,'myDAQ1','ai1','Voltage');
s.Rate = 50000;
s.DurationInSeconds = 0.1;
data = s.startForeground();
```

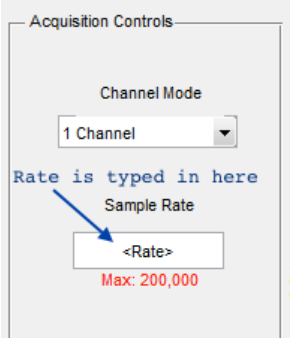
As we can see, the user creates a session object (`s`), for the vendor `'ni'` (national instruments). Then, two analog input channels, of variable type `'Voltage'` are added to said session object. Next the rate and duration are configured. Lastly, the session is started using the `startForeground();` command.

Note: there is also the ability to start a session in the background if other matlab code is to be executed while the session is running.

Now, it is easy to figure out how the GUI works, each time the user hits a button a 'callback' function is executed. These functions are designed to execute every time the user control is enabled.

For example, lets delve into how the rate is configured.

1. First, the GUI is started, then the user types into the Rate editable-textbox what rate they would like to configure their session to run at:



2. As soon as the rate is written the callback function is executed:

```
function Rate_Callback(hObject, eventdata, handles)
  textrate = get(handles.Rate,'string');
  ratedouble = str2double(textrate);
  handles.rate = ratedouble;
  guidata(hObject,handles);
```

   This bit of code shows how when the function is called, the preallocated value `handles.rate` is updated to the user-inputted value. Then, the global data structure is refreshed with the `guidata(...)` function. Data structures are used to pass this value to the acquire-button callback function.

3. Once the user presses the acquire button (after giving the GUI a duration), the Acquire callback function is called. This function reads as:

```
function Acquire_Callback(hObject, eventdata, handles)
  s_duplicate = handles.session;
  s_duplicate.DurationInSeconds = handles.duration;
  s_duplicate.Rate = handles.rate;
  data = s_duplicate.startForeground();
  handles.data = data;
  guidata(hObject,handles);
```

Here we see that the acquire button grabs the preallocated data acquisition session (which is initially configured in the startup function), sets the duration, sets the rate, then starts the session in the foreground. Additionally it saves the `data` to the `handles` structure and refreshes the global structure. We save the data to the structure in order to use the `csvwrite(...)` function when the save button is pressed.

This example is meant to show how the data acquisition functions have been interwoven into the GUI script. The underlying concept here is that of data structures, which can be passed from callback to callback in the background of the GUI.

6 Appendix II: Basic G-Coding

When programming in LabView, one does not write scripts (m-files), instead one creates a *virtual instrument*, or .vi-file. Virtual Images have two main parts, a *front panel* and a *back panel*. In the front panel, one configures user controls, be that graphs, indicators, sliders, virtual thermometers and gauges, etc. In the back panel, the programmer must use function blocks and other programming tools to string all the indicators/graphs to their data sources and so forth.

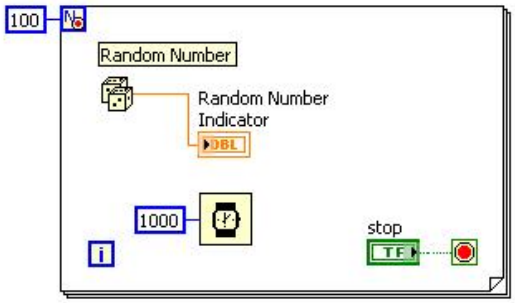
A LabView .vi is very much like the GUI described above in MatLab. In fact, a 'perfect' .vi would work like a GUI, wherein the user would never have to see the back panel. Unfortunately we do not live in a perfect world, and therefore as users it is important to understand how a LabView back panel operates.

Information, be that strings, doubles, integers, file paths, error codes, etc., flow around multicolored wires in the back panel, from block to block. Common coding structures like for loops, while loops, if-statements and so forth are represented as physical structures... here is a simple example to show how different graphical coding is compared to normal, textual language.

- **Textual:**

```
for i = 1:100
  display(rand);
  pause(1);
end
```

- **Graphical:**



It is easy read along with the textual language and see that it will display a random number 100 times, pausing for 1 second between each iteration. The graphical language is a bit harder to follow as it has no definite sequence. This is perhaps the hardest part about LabView's language, it is entirely atemporal.

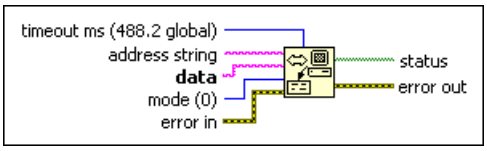
The blue '100' in the top left shows that the loop will run 100 times, the random number *primitive function* block will generate a random number and put said number into an indicator so the user can see it in the front panel, meanwhile the wait function will make the loop wait for 1000ms (1 second). One difference in the LabView code is that the user has the ability to toggle a button on the front panel to stop the loop (bottom right corner).

7 LabView Data Acquisition

LabView has, essentially, three types of sub-vi's, or 'blocks' to be used in data acquisition:

- **Primitive Functions**: These are the most basic form of blocks in LabView. They appear with a somewhat yellowish tint, and they generally perform more low-level computational tasks. e.g. 'build array', 'concatenate string', 'add', 'bundle'...etc. Data Acquisition was made possible using the 'GPIB Write' block in the program: "Grab Trace 480w'.vi'.

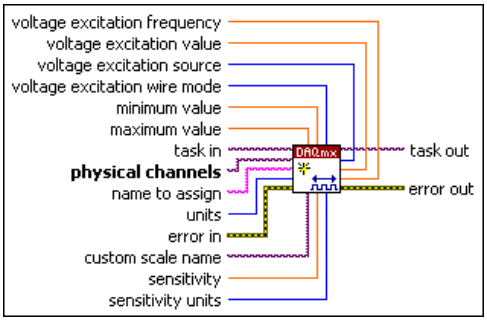
  Here is an example of a primitive data acquisition function:



Note: These Low-level blocks do NOT use the DAQ, they use the computer's own GPIB card to communicate with instruments.

- **Mid-Level Sub-VI's**: These are slightly more advanced blocks. These generally appear white and can perform more complicated tasks. examples of these include: 'Read/Write Spreadsheet', 'Numeric Integration'...etc. Data Acquisition is made possible using the mid-level Sub-Vi's under the 'Measurement I/O' tab, in the 'NI-DAQmx' menu. These functions can be used to control the DAQmx driver on a more intimate level than the next form of block.

  Here is an example of a mid-level data acquisition function, this one specifically is the 'create channel' block:

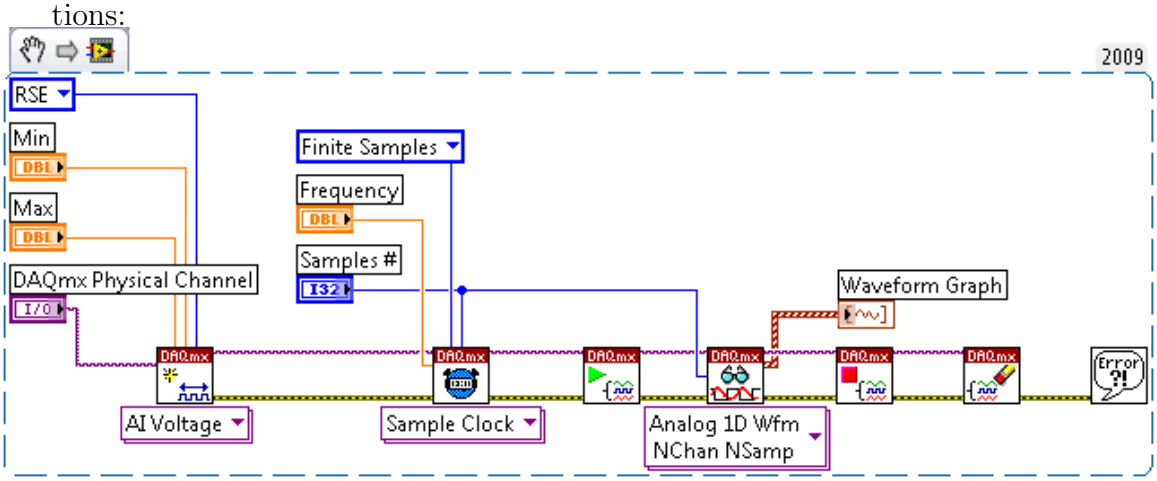


- **Express VI's**: These are by far the easiest blocks to use, but they allow for the least user-customization. In my opinion they are sort of a gimmick. These blocks are hardly blocks at all, they appear rectangular and pale-blue. The data acquisition express vi is called the 'DAQ Assistant'. When using the daq assistant, an entire new GUI opens up and the user is prompted to configure the data acquisition all at once... giving it a very limited range of flexibility. This express .vi works very well if the user only wants to output one type of signal, or only wants to collect data once and display it, for our higher-level lab needs the express .vi is not ideal.

  Here is what the express .vi for data acquisition (DAQ Assistant) looks like:

DAQ Assistant

Here is an example of a Data acquisition sequence using LabView functions:



This .vi would produce similar results as the GUI described in the MatLab section, the first channel creates a channel with user-inputted value for the physical channel, i.e. 'ai1' or 'ai0'. The next block configures the rate, number of samples to be collected and the sample mode (continuous or finite). The next block is the 'Start' block. Next comes 'DAQmx: Read', which deposits the collected data in a waveform graph. Then we have the 'Stop' and 'Clear' blocks.