

**DAQ User Manual**

Neuromuscular Control and Human Robotics Laboratory

Last Updated By: Tanner Bitz

Last Updated Dated: July 27th, 2018

**Hardware**

This data acquisition (DAQ) unit consists of a few essential elements. They are:

1. Teensy 3.6

- This the microcontroller that runs everything on the DAQ

2. DAQ127

- This piece of hardware takes 12 bit analog to digital measurements in any of it’s 8 channels. It also has an expansion connector (J2) on top that can be used for additional I2C device connections.

For more information about the DAQ127, the datasheet is located at:

<https://www.bipom.com/documents/peripherals/daq127_128_2543.pdf>

The microcontroller that runs the DAQ127 is the MAX127. It’s datasheet can be found at:

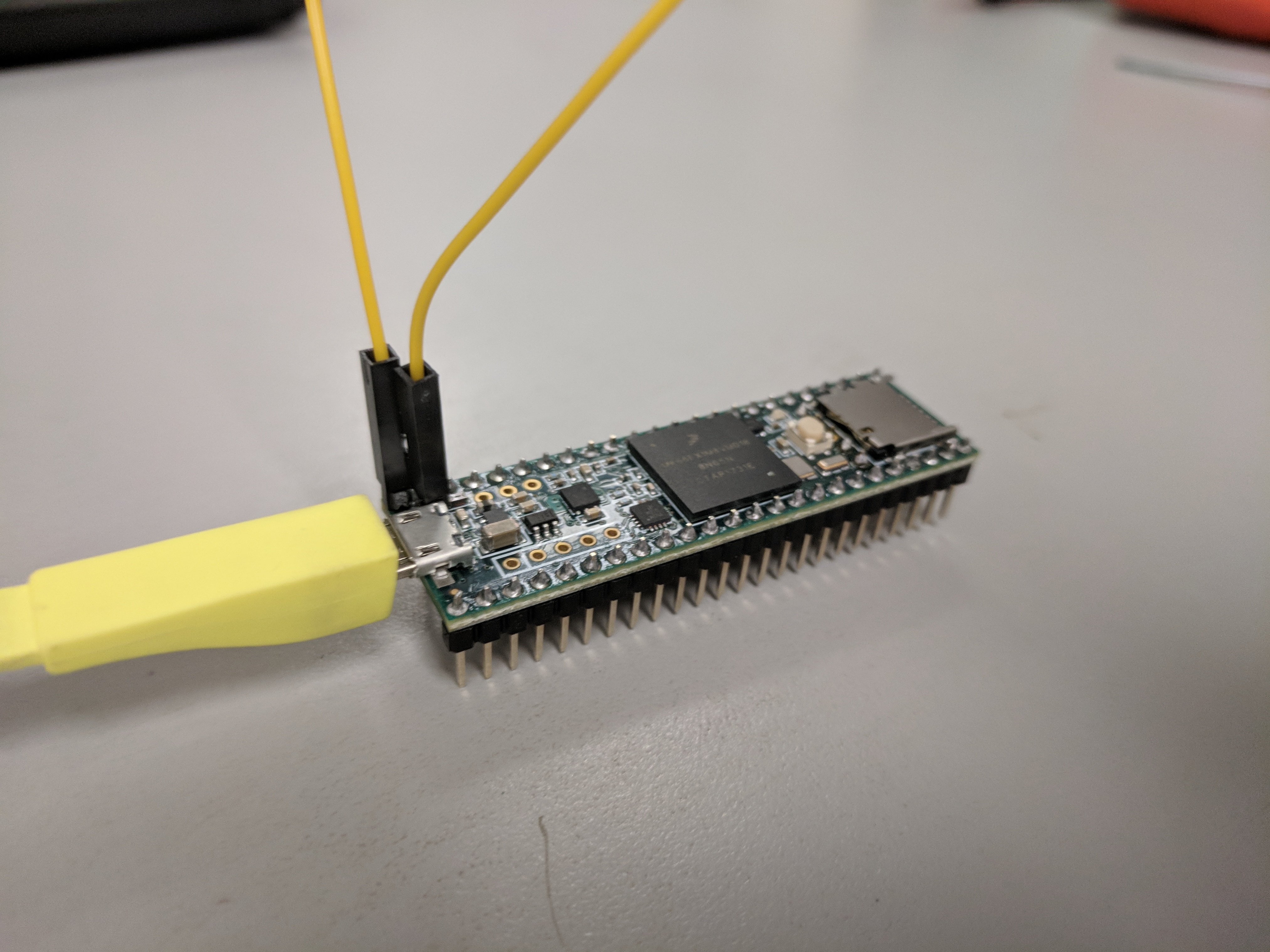
<https://datasheets.maximintegrated.com/en/ds/MAX127-MAX128.pdf>

3. Power Sources

This board must be supplied with 12V. This can be done using either of the power source connectors shown in the picture above.

4. Power Button

This must be pressed down to power the board after the power is hooked up to the power source connectors.

**Programming**

**To program the DAQ, the Teensy 3.6 must be removed from the board**. Be careful not to bend any of the pins during the removal process! This can be difficult sometimes as it is a very tight connection. Please be patient when removing it.

As of the writing of this document, this microcontroller is being programmed with the Arduino IDE. **To interface the Teensy 3.6 with the Arduino IDE, the program Teensyduino must be installed on your computer**. It can be found here: <https://www.pjrc.com/teensy/td_download.html>

Once Teensyduino is installed on your computer, the board should be listed in Arduino under the menu Tools→Board→Teensy 3.6

Next, **connect the 2 pins soldered on top of the Teensy with a jumper cable**.

Then **connect it to your computer with a micro USB cable**. The Teensyduino software should automatically open if the Teensy is connected, Arduino is open, and the board is chosen in the Arduino IDE. **Press the white button on the Teensy.**

**If you followed all these steps, it should now be in programming mode.**  To test whether it’s in programming mode, I suggest you try the Blink sketch in the Examples.

**Designed Use**

**Default Use Case**

By default, this DAQ will measure the first six channels of the DAQ127 (A0-A5) at 1000Hz. This DAQ reads at a 12 bit precision and is defaulted to read from -5 to +5V. When signaled by a serial command, this DAQ will start writing these readings in a comma separated format to a file on an SD microSD card on the Teensy3.6. This will continue until it receives a stop command from serial.

**Added Functionality**

Several other functionalities have been added to this DAQ to expand its use cases.

1. Change voltage range of individual DAQ channel to:

* 0 to 5V
* -5 to 5V
* 0 to 10V
* -10 to 10V

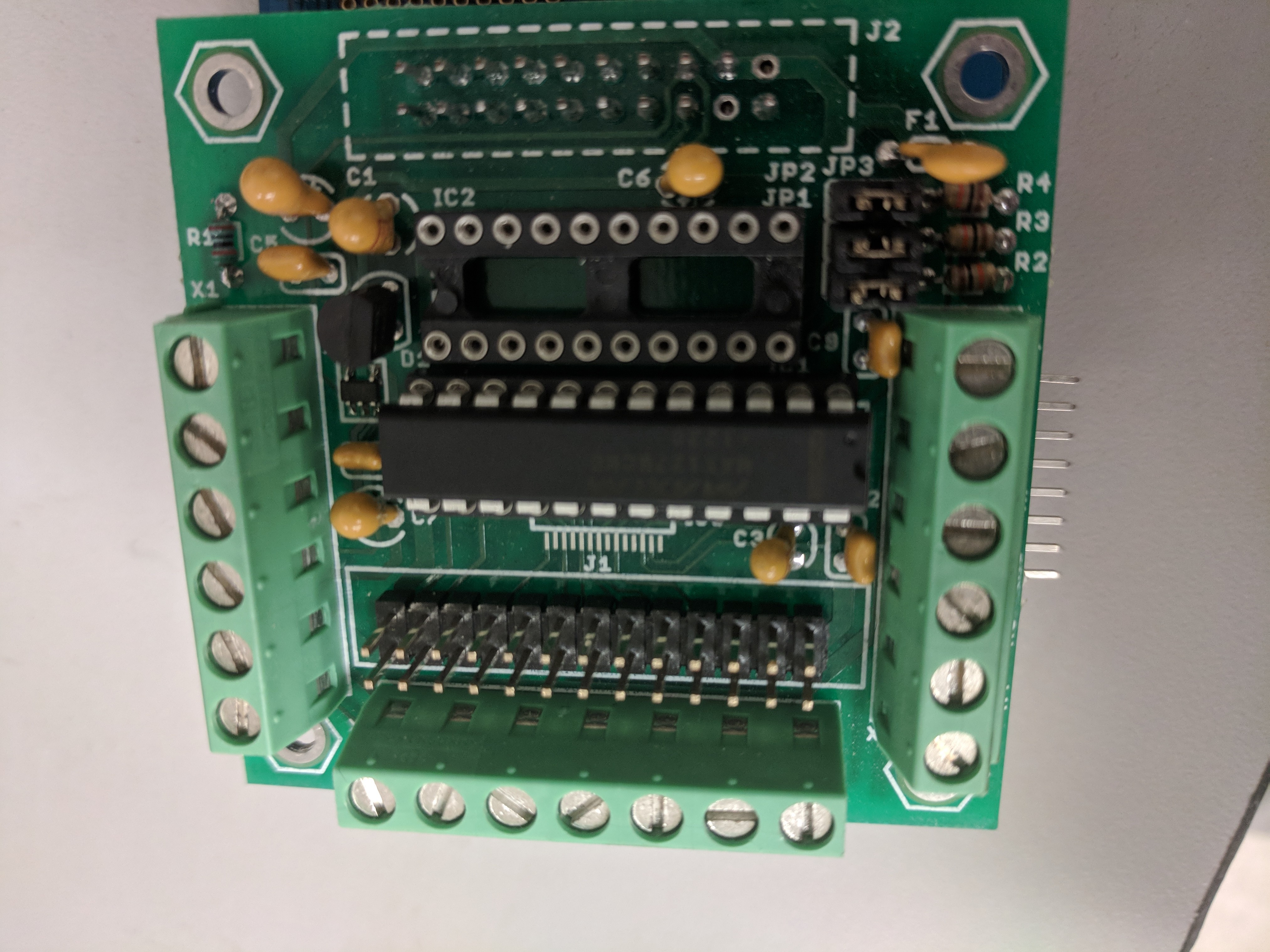
Use case: One of the signals you need to measure ranges from -8 to 8V. To avoid saturation in your readings, you will likely need to change that channel’s voltage range to -10 to 10V. Be mindful that when you make your measurement range larger, you are reducing your measurement resolution. It is always advised to use the smallest range that encapsulates your possible voltage ranges. To use this, please look at serial command “Change Voltage Range” and after you’ve sent the command to change the voltage range, be sure to check the DAQ did it with the “Print I2C Device Settings” serial command.

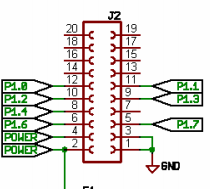
2. Customize I2C Device Address

Use Case: Read an external digital signal in sync with the analogs.

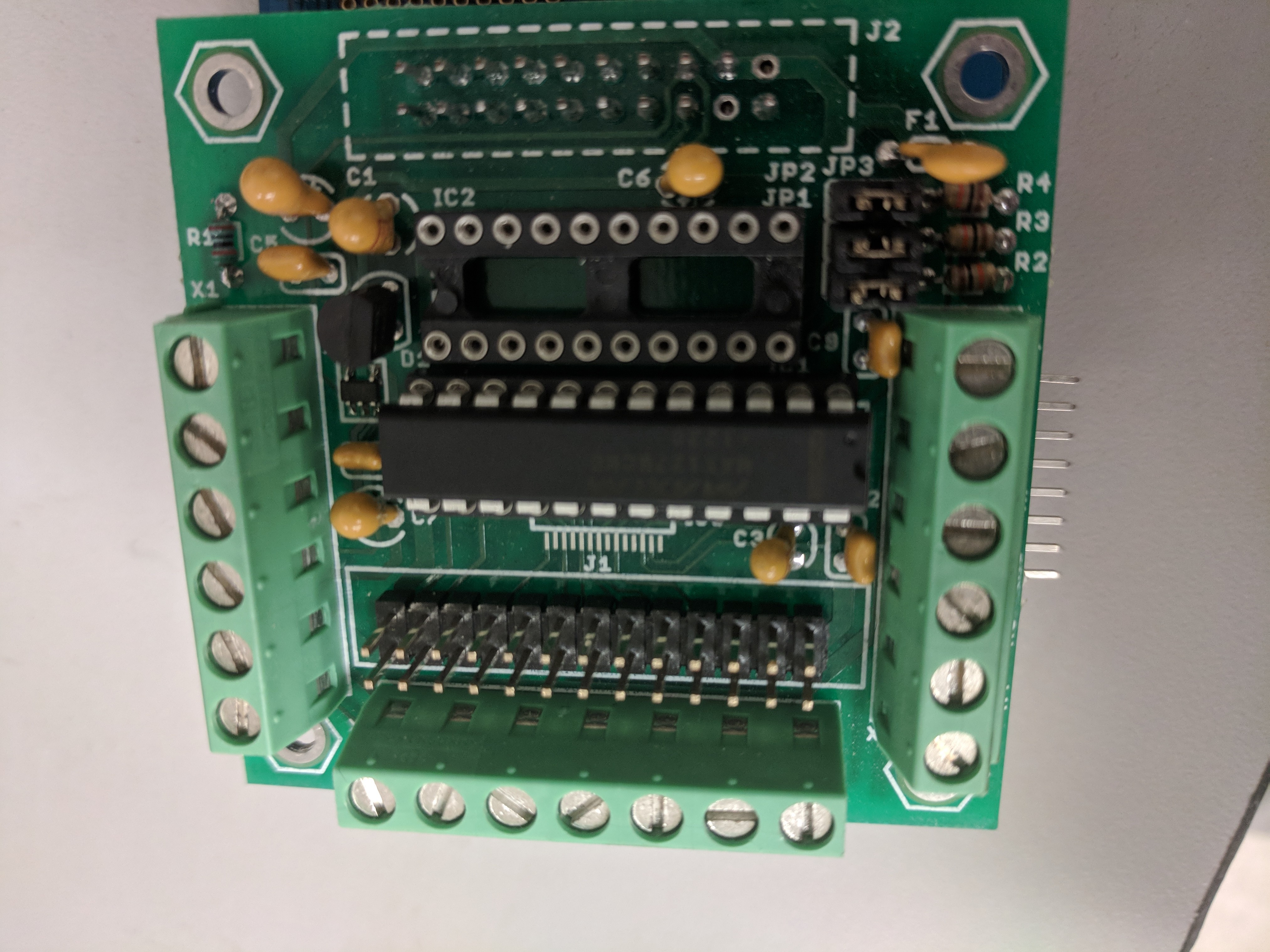
A little background: During each of the 1000 read cycles per second, the DAQ cycles through 6 I2C addresses and requests data from these addresses. When the microcontroller requests data from the DAQ127, the I2C device address is 0x28 and the control byte specifies the channel and voltage range. The DAQ127 then reads the signal and sends back 12 bits in 2 byte packages. (For the conscientious reader of this fine document, yes, that means DAQ sends 4 extra bits that we purposefully discard.) For more information about this, I encourage you to read the MAX127 datasheet listed at the beginning of this document.

Sometimes, we may want to read a digital signal in sync with our other measurements. To do this hook up your I2C device to SDA and SCL pins on the DAQ127 J2 connector. The SCL is on pin6 of the J2 connector and SDA is on pin 5.





**SDA (Pin 5)**

****

**SCL (Pin 6)**

Now that you have connected your I2C device, use the serial command “Customize I2C Device Settings” to change to the I2C device address you need. To check your I2C device was set up correctly, use the “Print I2C Device Settings” serial command. The DAQ will request 2 bytes of data during each call. It is up to you to decide how to use this functionality from here. I personally have a signal on an Arduino Mega that has a per-deteremined, period signal it cycles through and it passes the current value of the signal at each new call. If this doesn’t fit your needs, you may have to reprogram the DAQ to your liking.

3. Insert Value Into DAQ Readings

Use case: Insert a flag or value from the computer via serial to help with synchronizing signals in post processing.

Notice that we always read from 6 DAQ127 channels/external I2C devices, but we write 8 values to the files. Usually the 7th and 8th values (on what would be the 6th and 7th channel readings) are zero. This gives us a blank slate upon which to write values that act as a flag or anything you like. Use the serial command “Insert Value into DAQ Readings” to do this.

4. Print DAQ Readings

Use case: You need the DAQ readings for live plotting purposes or something while the DAQ is logging data. Use the “Print DAQ Readings” serial command to get an array of current DAQ values to process, plot, or do whatever you like with in close to real time.

**Serial Commands (presumably from a laptop/computer)**

All serial commands will be have a beginning character “<” and an ending character “>”. Please send commands without any spaces (though some work has been done to check and remove these). Note that since this is a serial protocol, all strings sent to the microcontroller must be byte strings.

**1. Start write to SD card**

Command String: <0,filename.txt>

Note: filename.txt can be any filename

**2. Stop write to SD card**

Command String: <1>

**3. Print DAQ Readings**

Command String: <2>

This command will return all readings over serial in the form:

<ch0read,ch1read,ch2read,ch3read,ch4read,ch5read,ch6read,ch7read>

where ch#read is an integer.

**4. Print I2C Device Settings**

Command String: <3>

This command will return all I2C Device Settings in the form:

<Channel: 0,DevAddr: 0x28,CntrlByte: 136,UseCntrlByte: True, VoltRng: -5V - +5V>

There will be 8 of these strings, so yes, the output will be 8x this long.

**5. Customize I2C Device Settings**

Command String: <4,I2C\_Device\_Address>

- I2C\_Device\_Address is the 8 bit unsigned integer value of the I2C device address

(i.e. valid range 0-255)

Note: This command would be used if you need to read a digital signal (probably from another microcontroller) in sync with other analog signals. Please read about this further under the “Added Functionality” section just above. The setup is purposefully restrictive for technical reasons.

**6. Change Voltage Range**

Command String: <5,CHANNEL\_NUM,VOLTAGE\_RANGE>

- CHANNEL\_NUM is an integer between 0 and 7

- VOLTAGE\_RANGE is an integer between 0-3

* 0 ------------ 0 to 5V
* 1 ------------ -5 to 5V
* 2 ------------ 0 to 10V
* 3 ------------ -10 to 10V

Example: <5,4,2> would change channel 4 to measure between 0 to 10V

**7. Insert Value into DAQ Readings**

Command String: <6,CHANNEL\_NUM,VALUE>

- CHANNEL\_NUM is an integer between 0 and 7

- VALUE is a 16 bit signed integer value to be written. (i.e. valid range –32768 to 32767)