There are 5 (1 optional) main steps to running arbitrary config files with the code and seeing waveforms. You must be running python 3 and have numpy installed.

1. Designate IP address for the board using Address Resolution Protocol (ARP)
2. (optional) Check that the board connects with check\_connections2.py
3. Populate config values in json\_blank\_config.py and run from command line to generate json file. It can be directly edited afterwards
4. Run data\_subscriber.py changing the field of hostname and the file name to the json config file from previous step.
5. Test\_plotter.py in command line with the generated data file and event length field edited to the event length of your acquisition (provided in the diagnostics during the run of data\_subscriber.py). Depending on save flags will possibly change what 16 bit words correspond to certain values. Also plots a single waveform.

**1.** Designate IP address for the board using Address Resolution Protocol (ARP)

1. Connect Ethernet and power 3316
   1. Set LAN IPv4 settings
      1. IP= 192.168.1.1
      2. Subnet mask = 255.255.255.0
2. Use the Address Resolution Protocol to associate an IP address with the 3316’s MAC address. The MAC address is 00:00:56:31:6X:XX for mac and 00-00-56-31-6X-XX for windows where X:XX is the last three digits of the printed serial number on the card in hex. For example with card with SN ending in 069 (in hex 0x45):
   1. arp -s “*IP-Address for 3316”* *“MAC Address”*
      1. I.E. arp -s 192.168.1.10 00-00-56-31-60-45
      2. Make sure running as admin (or use sudo)

**2**. Check that the board connects with check\_connections.py

1. In the common folder run in terminal:

python check\_connection.py “IP address” 1234

* 1. The number 1234 can be any port number you want, but that default one is not taken by any other software.
  2. Will print out to the terminal the card id (3316) and firmware version number if it is able to connect.

1. OK means successfully connected to board:



To Do: Update image since its now python check\_connection.py 192.168.1.41 1234 and because all cards are rev: 0x2010

**3**. Populate config values in json\_blank\_config.py and run from command line to generate json file. It can be directly edited afterwards.

1. Too much to describe here and will be written up more fully once confident/stable. However, most are explained in json\_blank\_config.py as comments. The not quite clear settings are the DAC offset. The table below gives good values:

Relevant Table of DAC Offset values from sis3316-m-1-1-v118.pdf:

A screenshot of a cell phone

Description automatically generated

1. As of right now whatever trigger threshold you set you should add 0x08000000 to the number (for some reason)
2. python json\_blank\_config.py

Edit the output filename here:

A close up of a screen

Description automatically generated

It also reads it into memory if run in an interactive IDE such as ipython to check the fields are as necessary. It’s just a dictionary.

**4**. Run data\_subscriber.py changing the field of hostname and the file name to the json config file from previous step.

1. Specifically change the hostnames to your card’s set IP address and config to the absolute path to your json file

A screenshot of a cell phone

Description automatically generated

1. If settings are right then it will save data based on the memory threshold set in the config file and/or for a set time per “generation” (similar to the 3302 code used in CCI2) and for a certain “max time” aka total acquisition time. If max\_time is not provided it defaults to 60 seconds. If gen\_time is not provided it defaults to the max. It will always switch based on either being full or based on the “Address threshold” flag set in the config file.



1. In main() the only value set is for 2 second total acquisition time.



1. python data\_subscriber.py
2. The output will show you all *set* values in the card by FPGA and channel number. If a readout occurs it will tell you how many bytes are transferred. It will also generate automatically a binary file with the data saved with a file name based on the current date and time.

**5**. Test\_plotter.py

1. The output of data\_subscriber will also dump the “event length” for each channel. Example snippet from below:

A screenshot of a cell phone

Description automatically generated

Here the event length is 306 (300 raw samples + 6 other data fields).

1. The Hit/Event Data is organized in the following way PER event:

A screenshot of a cell phone

Description automatically generated

The colored areas are the flags that can be set for certain hardware filtering functions. Initially these fields in the config file under “Hit Settings” can be set to zero and give only the first 2 lines. Setting “Save Raw Samples” to True will save the raw waveform. Struck uses 32 bits as their word but the acquisition code uses 16 bits instead (since that is what the much more numerous event data bits are). The script provided can provide an example of how to interpret various fields (the 0xE bit near the bottom is good to check things didn’t get indexed wrong somewhere) and plot single waveforms. Note that Struck uses little endianness but the numpy from file call takes care of that for you (it also means that the very first byte of an event is actually the format bits and channel ID NOT the timestamp bits 0-47). Bit shifting afterwards is necessary for getting the correct values.

1. Change event\_length in test\_plotter.py to your event\_length in step 1 of this part and the waveform number can give you the Xth waveform from your acquisition:

A close up of a logo

Description automatically generated

Note: A set frequency of 250 MHz means that every sample is every 4 nanoseconds.